DOCUMENT RESUME

ED 219 918 EC 150 008

AUTHOR Bates, Madeleine; Wilson, Kirk
TITLE ILIAD: Interactive Language Instruction Assistance for the Deaf. Final Report, September 1980-September 1981. Report No. 4771.
INSTITUTION Bolt, Beranek and Newman, Inc., Cambridge, Mass.
SPONS AGENCY Special Education Programs (ED/OSERS), Washington, DC.
PUB DATE [Sep 81]
NOTE 74p.

EDRS PRICE MF01/PC03 Plus Postage.
DESCRIPTORS *Computer Assisted Instruction; *Computer Programs; *Deafness; Elementary Secondary Education; Individualized Instruction; *Instructional Materials; *Language Acquisition; *Microcomputers; Program Descriptions; *Tutorial Programs
IDENTIFIERS *Interactive Language Instruction Assistance Deaf

ABSTRACT

The final report describes the syntactic, semantic, and tutorial components of the Interactive Language Instruction Assistance for the Deaf (ILIAD) system and the steps that have been taken to implement ILIAD on a microcomputer. It is explained that the ILIAD design allows for highly interactive tutorials in which the deaf learner specifies the content of each lesson; ILIAD in turn creates an individualized lesson according to the learner's specifications. Following an introductory chapter, chapter 2 surveys the linguistic and tutorial featurs of the ILIAD system with an overview of the philosophy and design underlying the tutorials and capsule summaries of various components of the ILIAD system. Chapter 3 describes the syntactic and semantic capabilities and the structure of the dictionary in the ILIAD system. A fourth chapter details several types of language tutorials, including those on sentence judgment, questions, subject-verb agreement, sentence patterns, politeness, and the "please" request. Various system development tools, such as the constraint mechanism, are reviewed in chapter 5. Microcomputer implementation of ILIAD is described in chapter 6 in terms of two subsystems--the utility section (which provides definitions and processing procedures for a variety of data structures used by the generation section) and the generation section (which corresponds to the generation part of ILIAD). Chapter 7 reviews dissemination of ILIAD, while the final chapter briefly considers the future of ILIAD. A list of sentences generated by ILIAD completes the report. (SW)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED219918

Report No. 4771

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

This document has been reproduced as received from the person or organization originating it.

Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

Final Report
ILIAD: Interactive Language Instruction Assistance for the Deaf
September 1980 - September 1981

Madeleine Bates

Kirk Wilson

Prepared by:

Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, MA 02138

Prepared for:

Office of Special Education
Department of Education
Grant and Procurement Management Division
400 Maryland Avenue, S.W., ROB-3, Room 5662
Washington, D.C. 20202

Attention: Dorland Bridgland
Grants Officer
[Dr. Allen Dittmann, Project Officer]

EC150008

Table of Contents

1. Summary 1

2. Overview 3

2.1 The Problem 3
2.2 ILIAD's Approach to a Solution 3
2.3 ILIAD Linguistics 5
2.4 ILIAD Language Tutorials 7
2.5 MicroILIAD 9

3. The Linguistics of ILIAD 13

3.1 Syntactic Capability 15
3.1.1 Base Component 15
3.1.2 Transformations 17
3.1.3 Control Capabilities 25
3.2 Semantic Capability and the Dictionary 29
3.2.1 Semantics 29
3.2.2 Dictionary 31
3.2.3 More Powerful Semantics 33

4. The Language Teaching of ILIAD 37

4.1 General Tutorial Features 37
4.1.1 Levels 37
4.1.2 Messages to the Student 39
4.1.3 Uniform Commands 39
4.1.4 Syntactic Choices 40
4.2 Sentence Judgement Tutorial 41
4.3 Question Tutorial 42
4.4 Subject-Verb Agreement Tutorial 45
4.5 Pattern Tutorial 47
4.6 Politeness Tutorial 49
4.7 Please Tutorial 52

5. System Development Tools 55

5.1 Constraint Mechanism 55
5.1.1 Relationships between Synspecs 55
5.1.2 Action of the Constraint Mechanism 56
5.1.3 Efficiency Considerations 57
5.2 The Syntactic Playground 57

6. Microcomputer Implementation 59

6.1 Utility Section 59
6.2 Generation Section 60
6.2.1 Base Component 60
6.2.2 Feature-Based Semantics 61
6.2.3 KLONE-Based Generation 62

6.2.4 Transformational Component 63
7. Dissemination 65
7.1 Reports and Presentations 65
7.2 ILIAD/Deafnet Project 66
8. Future of ILIAD 69
A. Sentences Generated by ILIAD 71

List of Figures

Figure 2-1: ILIAD Tutorials in Relation to the Generator 8
Figure 3-1: The Sentence Generator 13
Figure 3-2: Base Constraints and Their Values 16
Figure 3-3: Features in the X-Bar System 17
Figure 3-4: Sample Base Structure 18
Figure 3-5: Sample Transformation 19
Figure 3-6: Unmorphed Tree Structure 25
Figure 3-7: Morphed Tree Structure 26
Figure 3-8: Sample SynSpec 28
Figure 3-9: Sample Meta SynSpec 29
Figure 3-10: Portion of ILIAD Semantic Network 32

1. Summary

This report describes the syntactic, semantic and tutorial components of the ILIAD language instruction system and the steps that have been taken to implement ILIAD on a microcomputer. The goal of this work has been to design and implement an English language instruction system capable of *generating* a broad range of meaningful sentences as examples or exercises in tutorial lessons. This goal has been reached.

The ILIAD design allows for highly interactive tutorials in which the learner specifies the content of each lesson; ILIAD in turn *creates* an individualized lesson according to the learner's specifications. Because of the *generative* capability of ILIAD, the system continues to present examples or exercises for as long as the learner desires to study a particular grammatical or functional aspect of language use.

The prototype computer implementation of ILIAD has been written in InterLISP on a DEC System-20 computer at Bolt Beranek and Newman. Over the past two years a prototype microcomputer version of ILIAD has been designed and implemented. The microcomputer design has involved evaluation and modification of InterLISP ILIAD components for use in a UCSD-Pascal program. The microcomputer implementation, MicroILIAD, is now a functional language generation system with a subset of ILIAD syntactic capabilities and a flexible system of semantic representation. The MicroILIAD semantic system has allowed for a variety of tests and subsequent improvements of the original ILIAD semantics.

For two years ILIAD has been used by deaf children at the Boston School for the Deaf, by deaf children using a terminal in their homes, and by adults with access to the Deafnet computer communication system in the Boston area. Having deaf children and adults use and comment on prototype versions of ILIAD has been an important component in the design process of the ILIAD system.

2. Overview

This chapter surveys the linguistic and tutorial features of the ILIAD system. It provides a concise overview of the philosophy and design underlying the tutorials and capsule summaries of the various subcomponents of the ILIAD system.

2.1 The Problem

Recent technological developments have opened doors of opportunity for millions of handicapped people, yet many problems are exceedingly complex and require considerable innovation to reach even a partial solution. Foremost among these is the dilemma of language development in deaf children. Profound congenital deafness results in extreme delay in all aspects of language acquisition as well the formation of certain persistent deviant strategies for language comprehension and production.

Studies of deaf children indicate that English grammatical processes such as passivization, negation, question formation, relativization, complementation, and the formation of complex verb phrases pose particular problems for deaf students. Even students who appear to have mastered these constructions in isolation may be unable to combine them in a single utterance or to comprehend them when combined in complex sentences.

Hearing children learn their language while constantly immersed in it; they are surrounded by people and things (TV and radio) that use language almost incessantly. Deaf children, in contrast, are cut off from most sources of language and thus do not have the data from which to form hypotheses about how their language works. ILIAD is an attempt to increase the language data available to deaf children. It is an environment allowing a child (perhaps under parent or teacher supervision) to experience language in a manner that is carefully controlled though not limited. Once a tutorial has been started and tailored to the student's specific needs, ILIAD can generate an unlimited number of sentences for illustration and practice.

2.2 ILIAD's Approach to a Solution

The ILIAD Project has had as its goal the development of a generative computer system to aid deaf students in both the production and comprehension of written English sentences. The focus of the system is not on what to teach but rather on how to learn. For this reason, it is designed to be a

tutorial resource which is in large part under student control. Within certain limitations, the learner decides the content and basic features of each interactive tutorial session.

The basic characteristics of the ILIAD language tutorials are as follows: (1) the student has immediate control over the tutorial dialogue (i.e., the student can simplify a lesson or make it more complex according to his/her current level of language understanding); (2) examples of correct usage are provided using vocabulary within appropriate contexts and at the student's level of comprehension; and (3) the system is sufficiently motivating to encourage exploration of new areas of English with unlimited opportunity for continued practice on aspects of English not yet completely learned.

To provide sophisticated language tutorials meeting individual needs, ILIAD has been implemented using techniques of artificial intelligence. Its intelligence is in the form of *"knowledge"* about the structure of English along with *"skills"* to provide highly individualized tutorials focusing on different aspects of language use. The linguistic knowledge provides its capability to generate a broad range of simple to extremely complex English structures. The generative capacity is primarily syntactic, but ILIAD also knows a small amount about English semantics and the pragmatics (functions) of certain English sentence forms. With its knowledge of the English language, ILIAD has very powerful resources to individualize instruction in both grammatical and functional aspects of language use.

The ILIAD design has two flexible sub-systems, one specializing in linguistic capabilities and the other in tutorial interaction with the student. Given a set of sentence constraints (such as: past tense, negative, no relative clauses, and pronominalized subject), ILIAD will create an unlimited number of sentences meeting these exact specifications (such as: "It wasn't green", "She might not cry"). Such sentences are used to demonstrate grammatical or functional aspects of language use or as part of tutorial sessions to exercise the student's language skills. It is important to note that the goal of ILIAD is to help students learn *language*, not *grammar*.

ILIAD is also a linguistics research project, beyond its applications as a sophisticated computer program to meet a specific need. As in other artificial intelligence and computational linguistics projects around the country (principally in the Boston and Palo Alto areas), ILIAD has been developed using the InterLISP programming system which provides a very resourceful development environment for building and testing intelligent systems. ILIAD represents one of the first efforts to put intelligence into computer-based language instruction.

Once the prototype design was implemented in InterLISP and briefly tested, a conversion process was started to implement ILIAD on a microcomputer. The UCSD-Pascal implementation, MicroILIAD, has been started and a very preliminary version can now generate simple sentences. MicroILIAD takes advantage of some portions of the InterLISP system and in other cases, particularly where microprocessor capabilities are relatively limited, new designs have been developed.

The long range goal of ILIAD has been to become an "expert" language tutor. By expert we mean a system that has some knowledge of its domain, the English language, at least in a syntactic sense, and to some degree in terms of the functions and semantics of language. The system has been designed as a "linguistic playground" wherein ILIAD is not so much a teacher/director as a playmate/advisor. A successful student learns how to exercise the generative resources of ILIAD to explore a wide range of simple to complex language forms.

Throughout the design phase of ILIAD, constant effort was made to elicit and incorporate suggestions from deaf children and adults, teachers, parents and school administrators. While ILIAD does represent a sophisticated, intelligent system relative to existing CAI (Computer Aided Instruction) programs, prototype implementations have incorporated a wide range of comments from deaf children and adults who have used ILIAD and for whom the system is intended. Though ILIAD has some way to go before it is a field-tested language instruction system, it has demonstrated that a knowledge-based tutorial system can be successful and useful in the complex domain of language structure and function.

2.3 ILIAD Linguistics

The heart of ILIAD is a sentence generation component based on the linguistic paradigm of transformational grammar. This system is quite powerful and is capable of producing sentences ranging from simple, single clause sentences to complex structures which contain clausal complements of all types, such as, relative clauses ("The man who was angry chased Nan"), finite complements ("Dan said that Bette is intelligent") and infinitival complements ("Bill persuaded the girls to be nice").

More important than the range of sentences covered by this component is the way in which the sentences are represented. The sentence generator possesses detailed knowledge of the syntactic structure of each sentence it produces. For example, it maintains a record of the part of speech of

each word in a sentence (noun, verb, determiner, etc.), it notes the tense of each clause, and it is capable of determining the grammatical relations borne by different constituents; for example, which noun in a sentence is its subject, direct object or indirect object. In transformational terms, ILIAD produces phrase structure trees for both the deep and surface structures of each sentence.

Since ILIAD creates a detailed syntactic representation for each sentence that it generates, it is capable of generating not only a single sentence, but related sentences, as well. For example, if ILIAD generated the following sentence:

John ate the apple.

it could also generate related interrogative and pronominal forms, such as:

Did John eat the apple?
What did John eat?
Who ate the apple?
What did John do to the apple?
He ate it.

as well as other related forms not listed here.

The ability of ILIAD to generate related sentences allows it to help students realize which syntactic constructions are related. For example, the question "What did John eat?" may be generated by ILIAD alongside the declarative "John ate the apple", showing the student that both sentences are transitive (though this "meta-linguistic" terminology is avoided) and that they are related by a number of simple procedures which may interact to produce more complicated structures.

In addition to its ability to produce a wide range of sentences either singly or in groups of related forms, ILIAD is also capable of producing ungrammatical forms. Though this may not seem like an advantage, this feature of ILIAD is used in a diagnostic tutorial which tests the student's ability to distinguish grammatical utterances from ungrammatical ones (see Section 4.2). Moreover, the ungrammatical forms produced by ILIAD are not simply an ad hoc collection of random errors. Rather, these forms replicate the ungrammatical utterances observed in the actual writings of deaf children. Thus, this tutorial may be used to help students to realize that these forms are not acceptable as standard English. At the same time the teacher may be provided with diagnostic information about which ungrammatical forms students currently accept as correct.

Finally, ILIAD uses its representation of the syntactic structures of the sentences it generates to produce helpful tutorial information. For example, in a tutorial which requires the student to change the number of the subject (e.g. to make it singular if it was plural, or vice versa), ILIAD can inform the student what the subject of the sentence is and whether it is inflected regularly or irregularly.

2.4 ILIAD Language Tutorials

ILIAD tutorials focus on a a wide variety of language skills. The tutorial generates sentences relating to the skill it is designed to exercise and interacts with the student in a friendly way, eliciting responses from the student and gauging and monitoring those responses. The language tutorials allow the student to specify the difficulty level that the student wishes to work with and also allow the student to specify what type of sentences, within the framework of that tutorial's particular skill, the student would like to see. Figure 2-1 shows schematically the relationship between the tutorials and the language generator.

ILIAD currently has six tutorials, each of which addresses a different grammatical topic and a different area of capability in the ILIAD system. The tutorials, which are discussed in more depth in Chapter 4 below, are as follows:

- *Sentence Judgement:* In this tutorial the system presents sentences to the students and asks whether they are good or bad. This tutorial demonstrates that ILIAD can generate both standard and non-standard structures.

- *Subject-Verb Agreement:* This tutorial exercises the student's ability to change singular subjects to plural or vice-versa and to change the verb phrase to agree with the subject. This tutorial demonstrates that ILIAD can provide hints to the student based upon structural elements in the target sentence.

- *Question Formation:* This tutorial deals with yes-no questions, wh-subject question formation, and wh-object question formation. This tutorial demonstrates that multiple transformations may be specified for the exercise sentences generated by ILIAD.

- *Sentence Patterns:* This tutorial was requested by a teacher at the Boston School for the Deaf who uses basic sentence patterns in early language instruction. Students must be able to recognize up to five common sentence types. This tutorial demonstrated that a tutorial could be developed to meet a specific need, at the request of a teacher, in a short period of time (one week).

- *Politeness of Requests:* This tutorial focuses not on English grammar but rather on one of the linguistic skills needed to produce requests. It introduces the student to a wide

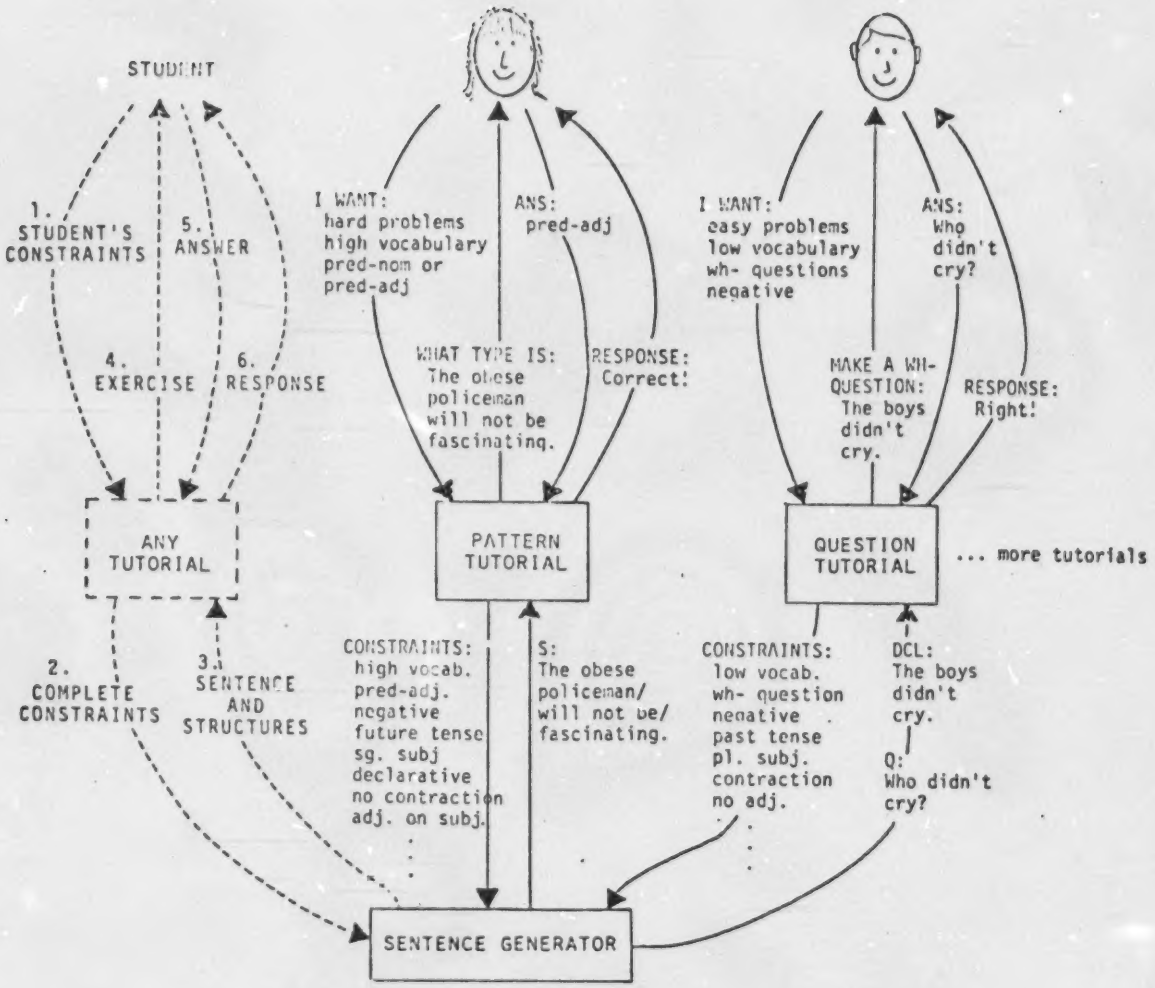
STUDENT
1. STUDENT'S CONSTRAINTS
5. ANSWER
4. EXERCISE
6. RESPONSE
ANY TUTORIAL
2. COMPLETE CONSTRAINTS
3. SENTENCE AND STRUCTURES
I WANT: hard problems high vocabulary pred-nom or pred-adj
ANS: pred-adj
WHAT TYPE IS: The obese policeman will not be fascinating.
RESPONSE: Correct!
PATTERN TUTORIAL
CONSTRAINTS: high vocab. pred-adj. negative future tense sg. subj declarative no contraction adj. on subj.
S: The obese policeman/ will not be/ fascinating.
I WANT: easy problems low vocabulary wh- questions negative
ANS: Who didn't cry?
MAKE A WH-QUESTION: The boys didn't cry.
RESPONSE: Right!
QUESTION TUTORIAL
... more tutorials
CONSTRAINTS: low vocab. wh- question negative past tense pl. subj. contraction no adj.
DCL: The boys didn't cry.
Q: Who didn't cry?
SENTENCE GENERATOR

Figure 2-1: ILIAD Tutorials in Relation to the Generator

variety of request forms from those which are strongly imperative to those which are very polite. ILIAD can generate over one hundred different sentence forms for a single request.

- *Use of Please in Requests:* This tutorial exercises the student's knowledge of the proper places to use PLEASE in a request. This tutorial utilizes a form of input which is convenient for students and produces a broad range of sentence forms that test their understanding of the use of PLEASE.

2.5 MicroILIAD

MicroILIAD is now a prototype language generation system running on a microcomputer. Its overall structure is similar to the ILIAD system implemented in InterLISP, but much of the design is different, due to the more limited computing power of the microprocessor and its memory constraints, and because the microcomputer implementation has been used to explore different types of semantic components.

The design of MicroILIAD has focused on achieving several goals of importance for a microcomputer-based system. These are:

- *Portability* — The system is written in UCSD-Pascal, a widely used microcomputer implementation of Pascal designed to reduce hardware and implementation dependencies. One version of MicroILIAD has been run without modification on an Apple II computer after development and compilation on a Z-80-based computer. Development work has been done on both the Z-80 and the Apple with relatively minor problems in compatibility of programs between the two.

- *Efficiency* — The internal structure of MicroILIAD relies extensively on the use of information coded in very compact ways (typically single bytes of storage) which can also be manipulated quickly. This has allowed the system to include all programs and data structures in memory without overlays and with room for expansion for additional grammatical capabilities, while achieving a sentence generation time for single clause sentences of 3-6 real time seconds (Z80 processor).

- *Modularity* — The basic processing in the sentence generation system is divided into separate modules for generation of the base tree, semantic analysis, and transformational processing (including morphology), so any basic element can be modified without altering the remainder of the system. There are also a number of utility modules for performing commonly needed processing and for maintaining various files of data structures.

- *Compatibility with InterLISP ILIAD* — Total compatibility with InterLISP ILIAD is not

consistent with the other design goals of MicroILIAD, but a substantial measure of compatibility has been achieved by the use of alternative data structures for system information. One data structure for a given type of information will be in InterLISP format, while another will be in a more compact internal representation, and programs are provided to convert automatically between the two data structures. This has allowed the InterLISP ILIAD transformations to be used in MicroILIAD in a form which is suitable for efficient processing.

The MicroILIAD system has been used to explore alternative semantic representations in order to determine strengths and weaknesses of these representations for sentence generation. The syntactic portion of MicroILIAD has been kept relatively simple compared to InterLISP ILIAD (with only single clause sentences and a small number of transformations) in order to allow more time to be devoted to this semantic exploration.

Three major semantic systems were investigated, and two were implemented. The first system explored was a semantic network representation like that of InterLISP ILIAD (see Section 3.2.1). While this representation is feasible for use on a microcomputer, the time and space required are quite large for the level of semantic analysis done and are not likely to be compatible with our efficiency goals, so this semantic network representation was not implemented.

In an effort to gain the effect of a network semantics with more efficiency, a feature-based semantic representation was implemented. In this system, each word had one or more fields of features (either features of the word itself or Katz & Fodor style selectional restrictions) represented as sets. Simple and fast set operations (union, intersection, and membership tests) could carry out the required semantic tests. This allowed an efficient implementation of a rough but serviceable semantics, but would have required a more elaborate language for representing semantic constraints not expressible as features (such as the complex requirements on gender and relationships between the subject and object of "kiss").

The feature semantics was implemented with vocabularies in two domains. In a limited playground vocabulary, a small set of features sufficed to rule out most anomalous sentences, producing sentences such as "The sports instructors catch an old soccer ball" and "The cheerful boy gives the bully a new football". A slightly expanded set of features was applied to a vocabulary chosen from newspaper stories, to give sentences with varying degrees of coherence, from "The astounding president escapes the assassination attempt", and "The angry Secret Service agent

shows the cameraman the brave movie star", to a not-so-successful "The uncomfortable psychiatric examination departs". Clearly, a more sophisticated semantics is necessary to cover this vocabulary range.

A more fundamental difficulty is present for both the network-based and feature-based semantics. Both are set up to rule out anomalous sentences (things we "can't say") but are not designed to create sentences based on things we have some reason to say. So we may get sentences such as "The cat is on the mat" and "Alicia saw the pen", both of which are grammatically and semantically correct but which, given no prior context, have no discernible relationship to each other or to a hypothetical situation where both might be true. In order to deal with this problem, a new semantic component was designed which is based on a representation of possible real-world situations in addition to general information, with sentences being generated which express some aspect of the situation. In this way, sentences can be generated which are related to each other because they each express aspects of the same situation.

This semantic system is based on the ideas of KLONE, a knowledge representation language developed over the past five years at Bolt Beranek and Newman. Use of the KLONE approach has allowed the semantics to represent not just hierarchical information as in a semantic network, but also relational information (e.g. properties of objects, participants in actions). This formalism distinguishes between general information (e.g a ball is a toy) and specific information (e.g Mary is sad). Generation proceeds from nodes representing specific information, so consistency is maintained, while the general information is used to give alternative expressions for the same situation. For example, we might generate "Mary threw the baseball", or from the same information produce the paraphrase "The sad girl threw the ball". (For a further discussion of the KLONE system, see Section 3.2.3.)

3. The Linguistics of ILIAD

The mechanism of transformational grammar was chosen for ILIAD because it offered both tight control over the surface syntactic form of a sentence and a good model for the production of groups of sentences that are syntactically related. Figure 3-1 shows the organization of the sentence generator.

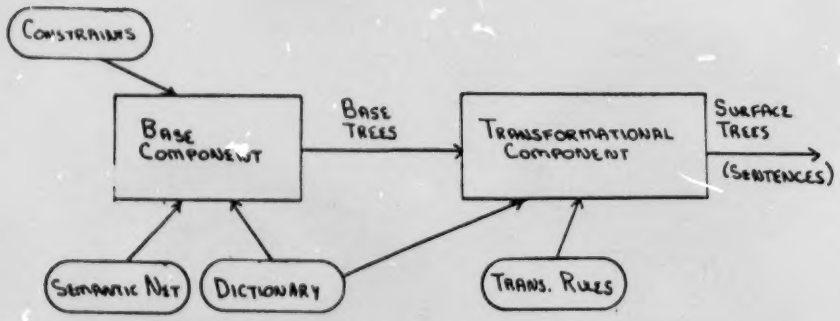

Figure 3-1: The Sentence Generator

Owing to its linguistic sophistication, ILIAD is able to produce a wide variety of English sentences. They range from commonly used constructions (transitive sentences, questions, passives, etc.) to expressions of great complexity, such as sentences with multiple question words ("Who gave what to whom?"), elaborate request forms ("Would it be possible for you to open the door, please?"), relative clauses ("The boy who baked some cookies was kissed by the girl who ate them."), etc. The following list presents just a few examples of sentences illustrating the many sentence types that can be generated (more examples are given in Appendix A).

Intransitive sentences:
The dogs are barking.
Bette giggled.

Transitive sentences:
The nice girl forgave the bullies.
The policeman closed the doors.

Transitive sentences with indirect object:
The teacher wrote the woman a letter.
The man gave a doll to Alicia.

Predicate adjective sentences:
Chris is happy.
The yellow apple is sweet.

Predicate nominal sentences:
Jake is a boy.
The doctor is a teacher.

Predicate adverb sentences:
The baseball is in the dirt.
The kids are on the slide.

Request sentences:
Tell the girls the story.
Let Bob tell the girls the story.

Tough-Movement sentences:
The girls were hard for Varda to ridicule.
The girls were hard to ridicule.

Comparative sentences:
The girl is hungrier than Bob.

Relative clause sentences:
The boy who was on the bike fell.
She saw the woman who baked that cake.

Want-Complement sentences:
Lyn wants the girls to hide.

Control-Complement sentences:
The teacher asked the boys to hurry. (Object Control)
Dianne promised to remain happy. (Subject Control)

The ability to generate ILIAD's total range of sentences requires 286 transformations. In

addition, 60 transformations were developed to generate sentences typically produced by deaf students; these transformations are currently used in the Judgement Tutorial (See Section 4.2).

Presently, the only major sentence types which are not generated are sentence conjunction, gerunds, and indirect questions. In the case of conjunction, we have deliberately refrained from adding the necessary base structures because of the semantic problems presented by conjoined sentences; for instance, the necessity of maintaining semantic relatedness between conjuncts (to prevent, for example, "Candy is sweet and John likes Mary"), the requirement that conjuncts appear in the correct temporal order ("John died and was buried." rather than "John was buried and died."), and so on, but projected changes to the semantic component will handle these correctly (see section 3.2.3).

3.1 Syntactic Capability

The sentence generator is composed of three major parts: a base component that produces base phrase structure trees, a transformer that applies the transformational rules to derive a surface structure tree, and a set of mechanisms to control the operation of the first two components. We will discuss each of these components separately.

3.1.1 Base Component

The base component produces the base tree which is associated with the sentence to be generated. The input to this component is a list of constraints which contain syntactic information about the type of sentence to generate (e.g. whether it is to be passive, have a relative clause on the subject, etc.) The output is a tree structure to which the transformations apply, to produce the desired sentence as well as related sentences (in the appropriate tutorials).

The base constraints which are the input to the base component determine the form of the base tree that is constructed. Most constraints take the value + or – ; others allow, in addition, a list of constraints, each of whose values is specified. A complete list of base constraints appears in Figure 3-2.

The output of the base component is a tree structure which represents syntactic categories as in the X-bar system of phrase structure. In this system, the major syntactic categories (N(oun), V(erb), A(djective) and P(reposition)) are treated as complex symbols which are decomposable into the features [±N] and [±V]. This yields the cross classification of these categories shown in Figure 3-3.

Name:	Value:
TNS	PRESENT, PAST, FUTURE, ANY, NOT-HABITUAL
SUBJECT	<a list of constraints indented below>
DIROBJ	– or <a list of the indented constraints>
INDOBJ	– or <a list of the indented constraints>
VERB	<a verb to be used as the head of the S>, NIL
PASSIVE	+, –
PROGRESSIVE	+, –, ANY
FOR-COMPL	<a list of sentence-level constraints>
THAT-COMPL	"
WANT-COMPL	"
CONTROL-COMPL	"
SUBJ-TOUGH	"
IMPERATIVE	2, 3, NIL
COMPARATIVE	– or <a list of constraints indented below>
TRANS	<a number or pointer to transf constraint list>
PRED-ADJ	+, –,<a word to be used as the adjective>
PRED-ADV	+,–
PRED-NOM	– or <list of constraints indented below>

{The following constraints apply within the context of a noun phrase.}

EMPTY	+, – (may be absent)
WH	+, – (may be absent)
NU	SG, PL, ANY
PER	1, 2, 3
DEF	+, –
ADJ	+, –, ?, <a word to be used as the adj>
NOUN	<a word which is to be the head noun>, NIL
PROPER	+ if NP must be a proper noun, – if can't be, NIL if it does not matter
REL-SUBJ	– or <a list of sentence-level constraints>
REL-OBJ	"
REL-INDOBJ	"
MASS	+, –, NIL
APPEAR	+, –, ?

Figure 3-2: Base Constraints and Their Values

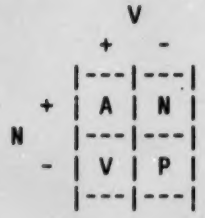

Figure 3-3: Features in the X-Bar System

The feature "N" marks a given category as "nounlike" (and thus corresponds to the traditional grammatical notion of "substantive") while "V" marks a category as "verblike". Nouns and adjectives are [+N] because they share certain properties (e.g. adjectives can be used in nominal contexts; in highly inflected languages, adjectives and nouns typically share the same inflectional paradigms, etc.) Adjectives and verbs are [+V] because they share (among other things) various morphological traits (e.g. certain verbal forms, such as participles, have adjectival properties). Verbs and prepositions are [–N] because they display common complement selection attributes (e.g. they both regularly take Nominal complements that bear Accusative Case).

In addition, each syntactic category contains a specification of its rank (given in terms of number of bars, hence the term "X-bar" system). For instance, a noun (N) is of rank 0 and is marked with no bars whereas the noun phrase which it heads is of the same category but different (higher) rank. Intermediate structures are also permitted; for instance, V′ (read "V bar") is that portion of the verb phrase which consists of a verb and its complements (e.g. direct and indirect objects, clausal complements, prepositional phrases, etc.) while V″ (read "V double bar") includes V′ as well as Auxiliary elements. For our purposes, we have adopted a uniform two-level structure across categories; that is, each category X is taken to have X″ as its highest rank, so that noun phrase (NP) in our system is N″, verb phrase is V″, etc. Minor categories (such as DET(erminer), AUX(iliary), NEG(ative), etc.) stand outside this system, as do S(entence) and S′ (a sort of super sentence, which contains S and clause introducing elements (or "subordinating conjunctions") such as *that*). These categories are not decomposable into the features [±N] and [±V], and, except for S and S′, they do not have different ranks. Figures 3-4 shows a representative tree structure.

3.1.2 Transformations

The transformational component modifies the base tree through the operation of certain rules

```
S'  [ TRANS.1 ]
  COMP  [ -WH ]
  S
    N''  [ +N -V PER.3 +DEF NU.SG GENDER.FEMALE +HUMAN +PROPER SOCSTAT.2 ]
      N'  [ +N -V PER.3 +DEF NU.SG GENDER.FEMALE +HUMAN +PROPER SOCSTAT.2 ]
        N  [ +N -V PER.3 +DEF NU.SG GENDER.FEMALE +HUMAN +PROPER SOCSTAT.2 ]
          Chris
    V''  [ -N +V -INDOBJ +TRANS ?PASSIVE ]
      AUX  [ -NEG ]
        TNS  [ +M +A -H ]
          past
      V'  [ -N +V -INDOBJ +TRANS ?PASSIVE ]
        V  [ -N +V -INDOBJ +TRANS ?PASSIVE ]
          steal
        N''  [ +N -V PER.3 +DEF NU.SG ]
          DET  [ +DEF NU.SG ]
            the
          N'  [ +N -V PER.3 +DEF NU.SG ]
            N  [ +N -V PER.3 +DEF NU.SG ]
              book
```

"Chris stole the book."

Figure 3-4: Sample Base Structure

(transformations) to create a representation of the surface syntactic form of the sentence (the final tree). Transformations are composed of three parts: a Structural Description, which is a pattern the tree must match if the transformation is to be applied, a Condition (optional) which must be fulfilled in addition to the Structural Description if the transformation is to be applied, and a Structural Change which describes the modification(s) to be made to the tree.

The Structural Change of each transformation consists of one or more functions, analogous to the transformational elementaries of traditional transformational theory. The operations that are performed by the rules are a combination of classic transformational operations (substitution, adjunction, deletion, insertion of non-lexical elements such as "there" and "do"), and operations that linguists sometimes relegate to the base or post-transformational processes (insertion of pronouns, morphing of inflected forms). By making these operations rule-specific, many related forms can be produced from the same base tree and the control mechanisms outside the generator itself can specify which forms are to be produced. The Condition of a transformation may be an arbitrarily complex Boolean expression or general LISP form which either describes relations among parts of

the tree matched by the structural description or specifies that some other transformation must have applied (or must not have applied). A sample transformation is shown in Figure 3-5.

```
SUBJECT-AUX-INVERSION

SD: (S' (FEATS (TRANS.1)) COMP (FEATS (WH.+))
     1                          2
         (S N'' TNS (OPT NODE (FEATS (M.+)))))
          3 4   5        6

SC:  (DeleteNode 6)
     (DeleteNode 5)
     (LChomsky 2 6)
     (LChomsky 2 5)

Condition: [NOT (EQ (QUOTE +)
                    (FeatureValue (QUOTE WH)
                                  (RootFeats 4]
```

Figure 3-5: Sample Transformation

The ILIAD system possesses a transformational component which can produce a wide variety of syntactic constructions. These include all of the most commonly used, simple sentence forms which the beginning student will need to learn, as well as some of the more complex forms which the advanced student might wish to become familiar with. Further, for the convenience of those working on the ILIAD project, all the transformations which are associated with a particular syntactic construction or with a range of related constructions are flagged with the same mnemonic prefix. (For example, those transformations responsible for the generation of questions all begin with the prefix WH-; those associated with relative clauses begin with RELATIVE-, and so on.) Thus, the transformations of the present ILIAD transformational component are broken up into several "families" of related transformations as well as several general "utility" transformations which are not associated with any particular syntactic construction but which enter into the derivations of various types of sentences. (Such "utility" transformations include NUMBER-AGREEMENT, AFFIX-HOPPING, various DO- insertion transformations, etc.) A complete listing of the transformations can be found in the report "ILIAD Data Base Reference" cited in Section 7.1.

Transformations are not only divided into families; the transformations, as a whole, are divided into two types of transformations: cyclic transformations and post cyclic transformations. The cyclic

transformations are further subdivided into those transformations which are used in the generation of grammatical sentences and "mangler" transformations which are used to generate ungrammatical utterances which are used in the Judgement Tutorial. Of the 346 transformations which exist now, 320 are cyclic transformations and 26 are post-cyclic, 260 of the cyclic transformations are used in the generation of grammatical sentences, and 60 are "mangler" transformations. We will detail the syntactic capabilities of each of these types of transformations separately.

The transformations which generate grammatical sentences are divided into the following families (in this and the following lists of families of transformations, each entry begins with the mnemonic flag associated with that family; the number in parentheses which follows this flag represents the number of transformations currently in that family):

- ADJECTIVE- transformations (4), which are used to form the comparative ("smarter", "more beautiful") and superlative ("sweetest", "most upset") degrees of adjectives.

- DATIVE- transformations (3), which allow for the alternation of indirect object phrases with and without "to" and "for" ("John gave Mary the book" ~ "John gave the book to Mary").

- GR- (= Grammatical Relations) transformations (11), which are used to find the subject, direct and indirect objects, main verb, and other relationally defined portions of a sentence, for use in the Help system.

- INF- (8) and INFERENCE- (3) transformations, which were designed for use in a projected inference tutorial.

- MODAL- transformations (14), which insert the various modal verbs: "can", "may", "must", "shall", "will" "is able to", "needs to", and "ought to".

- NEGATIVE- transformations (9), which include noun phrase negation ("no boys") as well as verb phrase negation ("The boy does not like candy").

- ONE- (2) transformations, which insert the pronominal element "one", as in "I like the green one".

- PASSIVE- transformations (5), which produce passive sentences, including "agentless" passives, such as "The girl was humiliated".

- PERFECTIVE- (2) and PROGRESSIVE- (1) transformations, which produce sentences with perfective ("The door has closed") and progressive ("The boy is running") aspect.

- PHRASE-BDRY- transformations (3), which are used to insert explicit phrase boundary markers in the tree, for use in the Pattern Tutorial.

- PLURALIZATION- (3) and SINGULARIZATION- (3) transformations, which are used in the Singular-Plural Tutorial.

- PRONOMINALIZATION- transformations (14), which pronominalize various noun phrase positions in a sentence, in first ("I am happy"), second ("The boy saw you"), and third ("The girl gave him a book") persons.

- Q- (=Quantifier) transformations (17), which insert quantifiers such as "some" and "every", as well as the numbers from one through ten.

- RELATIVE- transformations (9), which include all the restrictive relative clause possibilities; e.g. relative clause on subject, direct object, or indirect object, with the target of relativization either the subject or object of the relative clause. (They include transformations to produce infinitival relative clauses, such as "A book to read", as well, although the base structures for these forms do not exist at present). It is also possible to generate sentences with extraposed relative clauses; e.g. "The boy cried who was sad" from "The boy who was sad cried".

- REQUEST- transformations (72), which create various forms of requests, from stark imperatives (such as "Open the door!") to more elaborately polite requests ("I would appreciate it if you would open the door"). They are primarily used in the Request Tutorial.

- THERE-INSERTION, which produces sentences with existential "there" ("There was a boy on the slide", "There was a girl crying").

- THIS- (3) and THAT- (3) insertion transformations, which insert the demonstratives "this" and "that".

- TOUGH- transformations (2), which produce sentences such as "This rule is easy to ridicule", "It is easy to ridicule this rule", etc.

- WH- transformations (22), which produce yes-no questions ("Is the boy running?") as well as questions involving a WH-word (such as "who" or "what"; e.g. "Who is crying?", "What did you see?"). It is also possible to question quantified phrases, producing sentences such as "How many apples did John eat?" In addition, transformations exist to question genitive phrases (as in "whose book"), even though genitive phrases are not available from the base.

- WHIZ-DELETION- transformations (5) of various sorts, which relate relative clauses such as "The boy who was on the slide" with truncated forms such as "The boy on the slide".

The utility transformations include:

- AFFIX-HOPPING, which insures that the participles following the auxiliary verbs "have" and "be" are given the correct inflected form and that the person and number agreement marker in a clause is placed on the appropriate verb or modal.

- COMP/- (4) transformations, which insert subordinating conjunctions (or *Comp*lementizers) such as "that" and "for" into complement clauses.

- DET-SOME/ANY-SUPPLETION- (2) transformations, which guarantee that the correct form of the "polarity" item "some" appears in the appropriate contexts; e.g. as "any" after negation.

- DO- (4) transformations, which insert the dummy element "do" in the required contexts; e.g. in certain negative sentences ("Dianne doesn't like Chris") and in various questions ("Do you know the answer?").

- NUMBER-AGREEMENT, which causes the appropriate verb or modal in a clause to agree in person and number with the subject of that clause.

- PUNCTUATION- (5) transformations, which are used to insert commas in the appropriate places before and/or after "please" in various request forms.

- SUBJECT-AUX-INVERSION, which is used in the generation of yes-no and wh-questions.

Base structures can be created which permit the application of all these transformations, except as noted above. There also exist transformations which lack the appropriate base structures, although they are few. Such transformations have been tested on manually constructed base trees, so it is known that they function correctly. Such transformations include:

- CONJUNCTION- transformations (14), which allow for the production of various elliptical conjoined structures, including conjoined sentences with elided subject ("John likes Mary and hates Sue"), as well as more complex examples.

- EQUI, which generates sentences such as "John wants to go".

- EXTRAPOSITION-OF-P"-FROM-N", which produces sentences such as "A book came out by Chomsky" from "A book by Chomsky came out".

- INTRAPOSITION, which relates sentences such as "That Bill would do that seems strange" and "It seems strange that Bill would do that".

- IT-SPELL-OUT, which inserts expletive "it" in sentences such as "It seems to be raining".

- PARTICLE-SHIFT, which allows verbal "particles" to shift around the object of the verb, as in "He wrote the story down" from "He wrote down the story".

- RAISING, which produces sentences such as "John seems to be happy".

- REFLEXIVIZATION- transformations (3), which insert reflexives into various noun phrase positions in a sentence under identity; e.g. "John helped John" → "John helped himself".

In addition to the transformations which produce grammatical sentences, there also exist "mangler" transformations which deform the syntactic structures produced by the main set of transformations. These are of two types: those which produce ungrammatical sentences found in the writings of deaf students, (they are flagged with the suffix -/DEAF); and those which produce various deformations of grammatical structures which are not necessarily characteristic of deaf students (they are flagged with the suffix -**). At present there are 40 deaf transformations and 20 deforming transformations. The constructions for which deaf transformations exist are:

- BE- (5) and HAVE- (2) transformations, which duplicate the confusion and misuse of these two auxiliaries by deaf students. "Be" and "have" are deleted by some of these transformations, and, in some cases, are substituted for each other (e.g. "The boy has running", "The man is a coat").

- COMPLEMENT- transformations (8), which combine portions of infinitival and gerundive complements in the ways that deaf students do; e.g. "The teacher asked the boys to being nice".

- CONJUNCTION- transformations (4), which produce deformations of conjoined structures.

- LADV"- transformations (2), which confuse different types of adverbial phrases.

- N"- transformations (4), which perform various deletions and deformations on noun phrases.

- NEGATIVE- transformations (6), which substitute "no" for "not" and/or move the negative particle to positions where negation does not appear in standard English.

- OBJECT-DELETION, which produces examples such as "The woman got".

- PROGRESSIVE-DELETION, which produces examples such as "Jake was run' .

- RELATIVE-PRONOUN- transformations (3), which mimic various deaf relativization strategies: e.g. "The girl who Jake likes her is pretty".

- WH- transformations (3) and SUBJECT-AUX-REINVERSION, which are used to produce typical examples of deaf question formation.

Deforming transformations of the following type exist:

- -DELETION- transformations (10), which delete various constituents in a sentence.
- REQUEST-PLEASE- insertion transformations (3), which insert "please" in requests in positions where it should not occur.
- -SWITCH transformations (7), which scramble the order of constituents in a sentence.

Another type of transformation is the Post-Cyclic Transformation. Transformations of this type apply after all other transformations have applied and have as their domain the entire sentence (normally, transformations are limited to a single clause of a sentence). This type of transformation was created in order to allow the morphing process (which produces the inflected form of words whose root form appears in the tree) to be done by transformation. This allows ILIAD to retain information about the syntactic structure of each sentence generated. This information, in turn, has been used to construct Help and Hint mechanisms for various tutorials, to aid the students.

The post-cyclic transformations are divided into the following families:

- MORPH- transformations (14), which produce the correct morphological forms of the different parts of speech; e.g. the plural forms of nouns, the inflected forms of verbs, the correct forms of the perfective, progressive and passive participles, the comparative and superlative forms of adjectives, etc.
- PRONOMINALIZATION- -MARKING (4) and -SPELL-OUT (3) transformations, which mark the correct Cases on pronouns (e.g. NOM[inative]) on Subjects of finite clauses, ACC[usative] on Direct Objects, etc.) and produce the correct inflected forms of pronouns (e.g. "him" for the Accusative singular form of the third person masculine pronoun).
- RELATIVE-PRONOUN-SPELL-OUT- transformations (2), which produce the correct forms of the relative pronouns; i.e. "who" in relative clauses on a Human head noun and "what" otherwise.
- WH- SPELL-OUT transformations (3), which produce the correct forms of the WH question words; e.g. "who" or "what" for the interrogative pronouns which question noun phrase positions, as well as "which" and "what" (as in "which stories"), "how many" (as in "how many boys") and "how" (as in "how intelligent").

Figures 3-6 and 3-7 show the structure of a sentence before and after the application of the appropriate Post-Cyclic Transformations.

```
S'  [ TRANS.1 ]
  COMP  [ -WH ]
  S
    N''  [ +N -V PER.3 -DEF NU.SG +HUMAN GENDER.FEMALE SOCSTAT.2 ]
      DET  [ -DEF NU.SG ]
        the
      N'  [ +N -V PER.3 -DEF NU.SG +HUMAN GENDER.FEMALE SOCSTAT.2 ]
        N  [ +N -V PER.3 -DEF NU.SG +HUMAN GENDER.FEMALE SOCSTAT.2 ]
          girl
    V''  [ -N +V +PASSIVE +INDOBJ +TRANS ]
      AUX  [ ClauseNumber.1 -NEG ]
      V'  [ -N +V +PASSIVE +INDOBJ +TRANS ]
        V  [ -N +V +PASSIVE +INDOBJ +TRANS ]
          V  [ -N +V +PASSIVE +INDOBJ +TRANS ]
            ask@
          TNS  [ +H NU.SG PER.3 +M +A -H ]
            past
        N''  [ +N -V PER.3 +DEF NU.PL +HUMAN GENDER.MALE SOCSTAT.4 ]
          DET  [ +DEF NU.PL ]
            the
          N'  [ +N -V PER.3 +DEF NU.PL +HUMAN GENDER.MALE SOCSTAT.4 ]
            N  [ +N -V PER.3 +DEF NU.PL +HUMAN GENDER.MALE SOCSTAT.4 ]
              gentleman
        N''  [ +N -V PER.3 -DEF NU.SG ]
          DET  [ -DEF NU.SG ]
            a
          N'  [ +N -V PER.3 -DEF NU.SG ]
            N  [ +N -V PER.3 -DEF NU.SG ]
              question
```

Figure 3-6: Unmorphed Tree Structure

3.1.3 Control Capabilities

In order to manage the creation of the base trees and the application of the transformational rules, there are several layers of control mechanisms. The first of these is a set of *constraints* that direct the operation of the base component and indicate which transformations to try. The base constraints were discussed in Section 3.1.1; transformational constraints simply indicate which transformations are to be tried and which are to be ignored. There are a number of dependencies which exist among constraints. For example, if the transformational constraint for the passive transformation is turned on, then the base component must be instructed to produce a direct object and to choose a main verb that may be passivized; if the base constraint for a direct object is turned off, then the base constraint for an indirect object must be turned off as well. A data base of *implications* controls the application of constraints so that whenever a constraint is set (or turned off), the base and/or transformational constraints that its value implies are also set.

```
S'  [ TRANS.1 ]
  COMP  [ -WH ]
  S
    N''  [ DEF.NIL +N -V PER.3 -DEF NU.SG +HUMAN GENDER.FEMALE SOCSTAT.2 ]
      DET  [ -DEF NU.SG ]
        the
      N'  [ +N -V PER.3 -DEF NU.SG +HUMAN GENDER.FEMALE SOCSTAT.2 ]
        N  [ +N -V PER.3 -DEF NU.SG +HUMAN GENDER.FEMALE SOCSTAT.2 ]
          girl
    V''  [ -N +V +PASSIVE +INDOBJ +TRANS ]
      AUX  [ ClauseNumber.1 -NEG ]
      V'  [ -N +V +PASSIVE +INDOBJ +TRANS ]
        V  [ ROOT.ask -N +V +PASSIVE +INDOBJ +TRANS ]
          asked
        N''  [ +N -V PER.3 +DEF NU.PL +HUMAN GENDER.MALE SOCSTAT.4 ]
          DET  [ +DEF NU.PL ]
            the
          N'  [ +N -V PER.3 +DEF NU.PL +HUMAN GENDER.MALE SOCSTAT.4 ]
            N  [ -NU ROOT.gentleman +N -V PER.3 +DEF NU.PL +HUMAN
                 GENDER.MALE SOCSTAT.4 ]
              gentlemen
        N''  [ DEF.NIL +N -V PER.3 -DEF NU.SG ]
          DET  [ -DEF NU.SG ]
            a
          N'  [ +N -V PER.3 -DEF NU.SG ]
            N  [ +N -V PER.3 -DEF NU.SG ]
              question
```

Figure 3-7: Morphed Tree Structure

But constraints and implications are not sufficient to permit easy specification of syntactic constructions. The notion of "syntactic construction " transcends the distinction between base and transformational constraints and does not necessarily depend on their implications. One should be able to specify a syntactic construction such as passive or relative clause without having detailed knowledge of the constraints or their implications. In addition, one might want to request, say, a relative clause on the subject, without specifying whether the target of relativization is to be the subject or object of the embedded clause.

The ability of ILIAD to generate different syntactic constructions is greatly facilitated by a set of data structures called *synspecs* (an acronym for *Syn*tactic *Spec*ifications). Each synspec contains all the information necessary for the generation of a given form. This is useful because even a seemingly

simple construction may require the operation of several transformations, as well as particular base structure constraints. Each synspec contains a list of the appropriate base constraints and transformational constraints, the synspecs that could be used to generate a related incorrect sentence, and a description and examples for purposes of documentation, and other synspecs which are compatible or incompatible with it. When a synspec is actually "processed" during the generation of a sentence, the constraints take effect so that the desired form is produced.

Because each synspec is a "specialist" for a single construction synspecs are of great utility in both the Playground (see Section 5.2) and the tutorials. Since the number of syntactic constructions available in ILIAD now is quite large, it would be difficult for everyone to know all the base and transformational constraints for each construction, or even for a large subset of them. Synspecs, however, allow a user to generate any desired construction by simply invoking the appropriate synspec. Since synspecs bear mnemonic names, such as "passive-sentence", "transitive-sentence", "yes-no-question", etc., it is usually straightforward to find the synspec associated with a given construction. Moreover, each synspec also contain a description and/or example of the type of sentence it produces. Thus, in case the name of a synspec is ambiguous, it is easy to examine its description or example to determine exactly what sentences it produces.

Synspecs have also permitted the simplification of individual tutorials. This is due to the fact that various tutorials may generate the same constructions—yes-no-questions, passive sentences, various types of relative clauses, etc. Without synspecs, it would be necessary for each tutorial to contain the base and transformational constraints for every construction used in that tutorial. In the case of constructions which appear in several tutorials, this information would be redundantly listed in each of these tutorials. With synspecs, this information needs to be stated only once—in the appropriate synspec. An individual tutorial, then, need only know about those synspecs which produce the constructions it needs. Synspec names are also used when a student is specifying the kinds of sentences a tutorial should produce (see Section 5.1).

In addition to information about base and transformational constraints, synspecs also contain other types of information which may be useful in different tutorials. We have already mentioned that each synspec contains an example of the construction which it produces. Such Examples may be called by the Help system, in those tutorials which allow a student to select different construction types, when s/he is not certain what a given construction is. Various synspecs have a Question-for-

Student property which may also be used in such tutorials. There is also a Possible-Bad-Things-to-Do property, which associates with an individual synspec one or more synspecs which produce corresponding non-standard English sentences.

A sample synspec is shown in Figure 3-8; the complete list of 251 synspecs is given in the report "ILIAD Data Base Reference".

```
passive-sentence

   BaseConstraints :     ((PASSIVE . +))

   TransConstraints :    ((PASSIVE-AGENT-DELETION . -))

   Contradicts :         (intransitive-sentence
                                predicate-adjective-comparison-sentence
                                          predicate-adjective-sentence
                                          predicate-adverb-sentence
                                          predicate-nominal-sentence)

   Level:
                         1 DontUse
                         2 UserChoice
                         3 UserChoice
                         4 UserChoice

   QforStudent :         "passive sentences?"

   Examples :            ("John was punished by the teachers.")

   PossibleBadThings :   (be-deletion delete-by)
```

Figure 3-8: Sample SynSpec

Finally, it should be noted that there is a special type of synspec, the meta synspec, which, rather than setting up base and transformational constraints directly, can call other synspecs by interacting with the constraint mechanism (which is discussed in Section 5.1). This is useful in those instances where a construction has various subconstructions. For instance, the meta synspec *relative-clause* will randomly call either one of the synspecs *rel-clause-on-subject* or *rel-clause-on-object*. This synspec is useful, then, in cases where a relative clause is required, but it does not matter whether the relative clause appears on the subject or the object of a sentence. However, in those cases where a relative clause must appear in one position or the other, the latter two synspecs are available. Thus, meta synspecs allow for the creation of more general construction types,

whereas synspecs allow for the selection of specific sub-types of a given construction. An example of a meta synspec is given in Figure 3-9.

```
question

   Requires :          (yes-no-question wh-question)

   Contradicts :       (imperative declarative)

   Level:
                       1 UserChoice
                       2 UserChoice
                       3 UserChoice
                       4 UserChoice
```

Figure 3-9: Sample Meta SynSpec

3.2 Semantic Capability and the Dictionary

This section will describe the semantic component and the structure of the dictionary. Central to the capability of the semantic component is the organization of words in a hierarchical semantic network which will also be discussed.

3.2.1 Semantics

As the base component builds the base structure tree, it performs lexical insertion by picking words at random from the syntactic category needed in the tree and then using the semantic component to determine whether the selected word is semantically compatible with words chosen so far. In order to accomplish this task the semantic component is equipped with the following types of information:

1. The selected word together with its syntactic relation to the current situation.

2. A list of semantic constraints imposed by a higher syntactic structure which the chosen word has to satisfy. For example the word considered as a candidate for the direct object of the verb "eat" has to satisfy the constraints attached to "eat" under the direct object label. Those constraints indicate that the candidate word should belong to the semantic category of FOOD objects.

3. A list of registers which include syntactic and semantic description of the structure constructed so far.

In addition to the requirement of satisfying semantic constraints, the chosen word may cause to be set semantic registers which convey its meaning. Those registers are then checked against the current registers list to find out if they conflict with already existing registers. In a case of conflict the process is aborted (i.e., semantics rejects the word) and control returns to the syntactic component for another word choice; otherwise the new registers are added to the context and generation proceeds.

This mechanism provides the semantic component with a sophisticated way of referring to different levels in the base tree, and thus makes it possible to accept or reject a word on a more global basis than the semantic constraints permit. The following example will clarify the function of the registers: assuming the verb "wear" has been chosen as the main verb of a sentence, its subject is tested to verify that it belongs to the semantic category of HUMAN objects. The success of the test will invoke the setting of the subject's GENDER register because of its possible relevancy to the selection of the direct object. Then, if the word "skirt" is passed to the semantic component as a candidate for the direct object, semantics will attempt to set the register CLOTHES-FOR to FEMALE since skirts are intended for females only. This attempt will succeed only if the GENDER register of the subject is set to FEMALE, thus a sentence such as "The pretty girl wore a skirt" may be generated but "David wears a skirt" is rejected.

The semantic component is also aware of syntactic registers that might affect semantic considerations. Information about the NUMBER register of the subject and the direct object plays an important role when a verb like "ride" is under consideration. We do not encourage the generation of odd sentences such as "The boys have eaten the cookie" or "Debbie rides the bikes". Instead we generate "The boys have eaten the cookies" and "Debbie rides the bike". Determiner registers guides the semantics of predicate nominals: when both the subject and the predicate nominal are known to be indefinite, the subject will have to bear a class membership relation to the predicate nominal, as in "A ball is a toy" .

In order to handle the semantics of relative clauses efficiently, the semantic component includes a special mechanism for determining if a word chosen as a candidate for a verb in a lower clause is semantically compatible with the main verb in the upper clause. This is accomplished by a "backward" checking where all the semantic constraints of the "new" verb are checked against the corresponding registers in the registers list. If these new constraints violate any existing register

values, semantics rejects the verb. For example, in generating a relative subject on the subject for the sentence "The boy kissed Mary", the verb "cry" will be eliminated because the semantic constraints attached to its subject slot specify a negative emotion direction, while the emotion of the subject's register has already been set to positive when "kiss" was chosen.

The mechanisms described above provide the semantic component with powerful tools to deal with the semantics of the following sentence types: transitive and intransitive sentences, predicate adjectives, locative adverbs, want complements, that complements and sentences with relative clauses. The semantics of predicate nominals have been deeply investigated and have turned out not to be trivial at all (see the thesis *Representation of Nominal Concepts in Semantic Networks: Application for Generation of Predicate Nominal Sentences* by V. Shaked cited in Section 7.1).

3.2.2 Dictionary

The backbone of the dictionary structure is the semantic network which defines a hierarchical organization within the world of actions, objects and states. Figure 3-10 illustrates a partial semantic network. The semantic network consists of two components: concepts (IN UPPER CASE) and words (Capitalized). Concepts which are the nodes of the network represent sets of words that are closely semantically related. Each concept may point to its superconcepts and/or subconcepts. Words are the "leaves" of the network and have links to their appropriate concepts. For example, the words such as "boy" and "Bob" are linked to the concept "BOY" which further points to its superconcept "CHILD".

Currently, the dictionary contains about 112 different semantic classes (concepts), 114 verbs, 121 nouns, 37 adjectives and 9 locative adverbs as well as determiners, modals and other closed-class words. A complete listing of the dictionary may be found in "ILIAD Data Base Reference". Associated with each word are 5 kinds of information:

1. *Parts of speech*: A word may have more than one part of speech. For example, the word "kiss" is a verb and a noun.

2. *Features*: Syntactic, semantic, morphological and semantic "actions to be executed" are features of words which are processed as a word is added to a sentence. Syntactic features include such things as + or - passive and + or - transitive values. Morphological features show how the word is to be inflected. A link called EXAMPLE/OF indicates that a certain word is an instance of a more general concept. It distinguishes between the relation that words like "dog" or

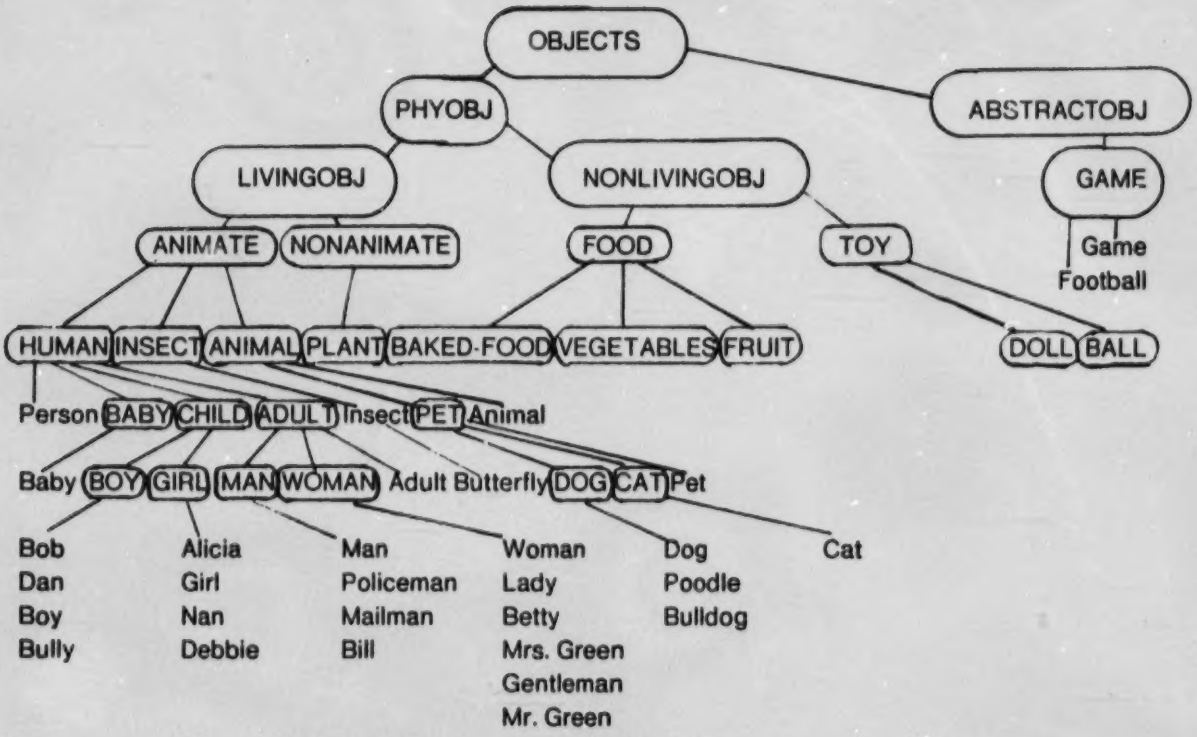

Figure 3-10: Portion of ILIAD Semantic Network

"cat" have to the "ANIMAL" concept, and the relation that the word "animal" bears to the same concept. This link has been introduced mainly to assist the semantics of predicate nominals.

3. *Constraints*: Semantic constraints are specified for the various syntactic case slots that a word might have. Much effort has been put in expanding this kind of information, and syntactic constraints were taken into consideration too. In the event that semantic information applies to a group of words (i.e. concepts) it is attached to the concept and is inherited by the words. The fact that nouns such as "boy", "man", "girl", can be modified by emotional adjectives, is attached to the concept "HUMAN". This way the necessary semantic information is stated only once instead of repeating it for each word separately.

4. *Complexity*: The difficulty level of words can be specified, e.g., Level 1 words would include "play" and "happy" while Level 4 would include more sophisticated words such as "humiliate" and "edible".

5. *Vocabulary Type*: The main ILIAD vocabulary is a "Playground" vocabulary focusing on persons, animals, objects, actions and locations associated with a children's playground. The purpose of multiple vocabularies is to allow the student

> to change the topics in the tutorials simply by changing to another vocabulary type; changing vocabulary type, however, will have no effect on other syntactic or semantic constraints which have previously been set. ILIAD will make it possible to focus on grammatical aspects of English in a variety of contexts.

As the semantic network expanded and more information was added to the dictionary, we realized the need for a routine to control errors that might occur during the input of new entries to the dictionary. This word definition routine minimizes the amount of information typed by the user, and interacts with him/her through a prompting process. The dictionary maintenance functions are used when words are edited, and it helps keep track of which dictionary files need to be remade. This mechanism has made it possible for some people who were not programmers or developers of the ILIAD system to enter new vocabulary with a minimum of training.

3.2.3 More Powerful Semantics

The semantic network described above constrains the sentence generation of ILIAD so that anomalous sentences are avoided. But a problem arises when more than one clause is generated, whether in a single sentence (e.g. a relative clause) or in a sequence of sentences. The clauses are essentially independently generated (although clauses within a single sentence must satisfy inter-clausal constraints), so they are not necessarily semantically related. A sequence of such clauses can seem non-coherent to the ILIAD user even when each sentence is acceptable itself.

Another problem with simple network semantics arises because the semantics operate essentially as a filter. Out of the universe of syntactically acceptable strings, it rules out only those which are strongly semantically anomalous. This can cause a problem with efficiency, since backup may be needed before a semantically acceptable string is found, and it can also generate subtly peculiar sentences such as "The fire chief writes the story" which is certainly not anomalous, but which is a bit unmotivated.

To deal with these problems, a new semantic system was designed which allows representation of more information, including both general semantic knowledge and representation of specific real-world situations. This system is based on the knowledge representation language KLONE, with both deletions of substantial portions of the present KLONE system and a few extensions appropriate to the language generation domain.

The KLONE semantic system uses the basic constructs of CONCEPT (essentially a structured

object), SUPERC (for hierarchical relations), and ROLE (for arbitrary structured relations). Syntactic processing is facilitated by the addition of SYNCONCEPT (syntactic concept) nodes and POS (part of speech) links. While these syntactic types could have been included as instances of other KLONE nodes and links, their separation from the semantic information partitions the network and reduces the size of the search space when accessing network information.

The problem with unmotivated sentences is dealt with by the ROLE links. Each ROLE specifies a relation between objects or classes of objects (e.g. A person has a height, or Jim is 6 feet tall). These ROLEs contain exactly the information which we are likely to want to generate, so we can search the list of ROLEs to determine candidates for generation rather than picking random words and then checking for anomalies. The semantic system can also mark the information associated with a ROLE as obligatory or optional and may indicate how this information is to be expressed (e.g. an agent of an action might be marked as obligatory, expressed as grammatical subject). It may even mark the information as known but not usually expressed (e.g. every person has a head).

The problem with multiple clauses is also related to the lack of sufficient information in ordinary semantic networks. ROLE information is one type of missing information, but more important for multiple clause coherency is the notion of specific (real-world) information. KLONE distinguishes between generic and individual concepts, and between role definitions and role fillers (essentially generic and individual roles). The KLONE semantics uses this distinction to separate general information (e.g. a boy is a child) from specific information (e.g. Fred is tall). A collection of specific information (e.g. a set of facts making up a description, or a set of actions making a story) is called an instance. Sentence generation is done using the information making up the instance, so that each sentence will be related to the same set of facts, and the set of sentences will exhibit greater coherence.

An instance describing a particular situation can be hand-coded, but in this case the sentence generation would be limited to exactly the participants and specific actions coded, reducing the flexibility of generation which is a major feature of the ILIAD system. To extend the generation range, a new level of description, called the scenario, has been included in the KLONE semantic design. A scenario is a structured collection of information which includes generic as well as specific facts. For example, an instance might be [Joe chased Mary. Mary cried.], while a related scenario could be [A bully did a negative action to a person. That person cried.] Using scenarios, a wide variety of

sentences can be generated, while still maintaining the coherence resulting from the use of instances. This is achieved by instantiating a scenario and using the resulting instance for generating a set of sentences. When an unrelated set of sentences is to be generated, a different instantiation can be used. This instantiation is done by combining the information from the scenario (e.g. A bully did a negative action) with specific and general information coded in the network (e.g. Joe is a bully. To chase is a negative action.)

The use of specific instances for generation also helps with some problems relating to reference. In the network-based semantic system, choice of definite or indefinite articles is made randomly, and pronominalization is triggered by use of the same word a second time. In the KLONE semantics, each word has a referent in the KLONE network, and the referents can be marked according to whether they have been expressed, so an indefinite article is generated when an object not previously mentioned is used, and a definite article when the object is marked as previously expressed. In a similar way, pronominalization could be triggered by identity of referent rather than identity of word.

Many of the ideas for the KLONE semantics have been implemented in the MicroILIAD system. A small KLONE network was used, containing both general and specific information, and sentence generation demonstrated the ability to use the general information to create a variety of sentence forms to describe a single action, as well as the use of reference markers to handle definite and indefinite articles. The major part of the design which has not been implemented is the use of scenarios and instantiation, as all present MicroILIAD instances have been hand-coded.

4. The Language Teaching of ILIAD

The language tutorials mediate between the sentence generating mechanism and the student. Thus, after the student makes choices regarding sentence complexity, a "prescription" is passed to the sentence generating mechanism which determines the complexity of sentences to be generated for a particular tutorial. A fundamental principle of ILIAD tutorials is to provide the student with as much control as possible over the content of each tutorial session. This requires a control mechanism which allows the student to make all sorts of specifications as to the type of sentences to be generated, without falling into the trap of making invalid or contradictory specifications. This mechanism takes the burden off the student of understanding precisely what each type of specification entails or prohibits and in so doing frees the student to experiment and try out new sentence types. This, in turn, leads to greater variety in the tutorials and greater student interest as well as a fuller use of the system.

ILIAD has several language tutorials which are described below in detail. The examples given were all produced in actual use, though the interactions have been slightly edited for brevity and clarity.

4.1 General Tutorial Features

The tutorials are all designed to to appear as uniform as possible to the student so that the interaction remains familiar no matter which tutorial is being used. This section discusses those features which are common to all the tutorials.

4.1.1 Levels

The tutorials have four different levels of difficulty, which the student can choose among. As the difficulty level increases the range of possible sentence types increases and this in turn allows the student a greater variety of sentence types to work with. For instance, sentences involving relative clauses are only presented at the higher levels.

In addition to the difficulty level, the student can also specify one of four vocabulary levels, ranging from very simple words to those which a deaf student might not be expected to know even at the high school level. This governs the types of words that the tutorial will make use of in the sentences it produces and in the comments it makes to the student.

These two level specifications give the ILIAD tutorials the potential to be used by a widely differing range of students from the very naive to the quite sophisticated. In addition, as a student progresses and begins to master one level s/he can simply increase the level of the tutorial and be freshly stimulated with more complicated sentences. The ability to control the vocabulary level independently of the syntactic level makes it possible for a student to experiment with these variables in order to find the combination that s/he is most comfortable with, as well as permitting the overall complexity of the sentences produced to be increased very gradually.

The following transcript shows a student's interaction with ILIAD.

```
                    ILIAD - Version 5

What is your name?
>jack

Thanks, Jack.
```

[The student's name is remembered for later use. Now the set of possible tutorials is shown.]

```
You can...
1.  decide whether sentences are good or bad
2.  play with singular and plural sentences
3.  make questions
4.  match sentence patterns
5.  learn how to use Please
6.  learn about polite requests

Do you want 1, 2, 3, 4, 5 or 6?
>3

You can see...
1.  short sentences
2.  medium sentences
3.  long sentences
4.  very long sentences

Do you want  1, 2, 3  or 4 ?
>3
```

[This is the syntactic level choice.]

```
The sentences can use...
1.  very easy words
2.  easy words
3.  medium words
4.  hard words

Do you want  1,  2,  3,  or  4 ?
>4
```

[This is the vocabulary level choice. At this point, ILIAD branches to the particular tutorial that was chosen.]

4.1.2 Messages to the Student

Congratulatory messages are printed in response to a correct answer by the student. The exact message printed depends on the level that the student is working at and sometimes includes the student's name. This is intended to make the tutorials friendly and to hold the student's interest so that a sophisticated student is rewarded with stimulating and unusual messages. On the other hand, the messages given in response to a wrong answer are short and direct so as not to reinforce incorrect answers; these too vary depending on the syntactic level selected by the student.

4.1.3 Uniform Commands

To make the ILIAD system as easy as possible to use for students who are not necessarily familiar with computers, it was decided to make the entire student interaction uniform and homogeneous in the sense that all points of interaction will have the same request format and standard options for student input. This was achieved by having one function that controls the student interaction. This function can have many different types of arguments so that it can be made to expect one of three types of input from the student: (1) a sentence, (2) one word, (3) some option from an options list.

The uniformity is in the standard options that can always be typed by the student to the ">" prompt. Thus the student has to know only these few standard options to be able to use the system. In fact, the student has only to remember one, the question mark, since typing that will show all the other available options.

The standard options which are always available are to end the lesson, to send a message to the system developers (this feature has been very useful for conveying suggestions and bug reports

to the programmers responsible for maintaining the system), to change the student's choices about the kinds of sentences to be produced, to repeat the most recent question, and to get help or a hint.

If the student uses the Help option, s/he is given a message that provides some useful information relevant to that particular point in the system. For example, when the system asks the student if s/he wants to see negative sentences and the student not knowing what is meant by negative sentences, types Help to find out, some examples of negative sentences are given and then the > prompt is given again:

```
Do you want to see ...
negative sentences?
>Help

Here are some examples to help you.
The apple isn't good.
John did not see Mary.
>
```

At this point, Help messages have not been built into all parts of the system. However, the Help mechanism has been set up so that adding those various messages is now a simple matter.

If the student chooses the Hint option, s/he is given a hint relevant to that particular point in the system. This is similar to the Help option above, but can be more specific.

Since the ILIAD system has extensive "knowledge" of syntax which it uses to build a sentence, it can use this information to further instruct the student. The Help and Hint mechanisms do just that. Thus, they can provide information as to the various syntactic parts of a sentence, such as what the subject or object is, or information concerning the verb and how to change its tense. This use of the Help/Hint mechanism will be illustrated in the tutorial examples below.

4.1.4 Syntactic Choices

The synspec constraint mechanism allows the tutorials to make full use of the wide variety of sentence types that the sentence generating component can generate. The constraint mechanism was necessary to ensure that only sentences desired by the student and applicable to each tutorial were generated. The constraint mechanism allows the student to specify various choices to control the types of sentences that are generated within each tutorial. The student is presented with a set of choices of syntactic structures the elements of which are determined by the syntactic level s/he has

already chosen. The student can choose any number of these structures, and can refuse any number of them as well. As the choices are made, the set of subsequent choices expands and contracts so that at all times ILIAD displays only choices which are compatible with the tutorial, the student's previous choices, and the current level of syntactic complexity. The choices made by the student remain in effect for the duration of the tutorial, but before each sentence is generated the tutorial makes further random selections to produce a variety of structures in the exercises. Additional explanation of the constraint mechanism is given in Section 5.1.

4.2 Sentence Judgement Tutorial

The *judgement tutorial* generates sentences of various syntactic types and requires the student to judge whether the sentences are grammatical or not. The major syntactic constructions used in the tutorial include negation, yes-no questions, wh-questions, and relative clauses on the subject and object; the use of these constructions is governed both by the syntactic level chosen by the student and by requests for specific types of sentences.

The judgment tutorial shows the full power of using a transformational model in that it is possible to generate so-called "Deaf English" sentences. This means that the tutorial can present sentences that are incorrect in a way that is typical of the types of mistakes that deaf students make.

An example of an interaction in this tutorial is the following:

```
**************************************************************************

SYNTAX TUTORIAL - you will decide whether sentences are good or bad

Now you will choose the kind of sentences you want.
Type choices to see additional choices; type go when finished.
```

[The constraint choice mechanism illustrated here was implemented in the last six months of the project and an appropriate student interface has not been developed. The system appropriate for student use would present examples to communicate choices rather than a list of linguistic terms.]

```
declarative                          imperative
question                             predicate-adjective-sentence
predicate-nominal-sentence           predicate-adverb-sentence
negative-sentence                    relative-clause
>>declarative
>>go
```

```
Ok.  Now we will start!

1.  The bullies bought Mary a baseball.
Is that sentence good or bad, Jack?
>...
I can't wait any longer, Jack.
```

[The tutorial is designed to wait a certain amount of time, at which point, if the student has not responded at all, the correct answer is given and the next exercise is presented.]

```
You can say
            The bullies bought Mary a baseball.

2.  The child stays on the bike.
Good or bad, Jack?
>good
Absolutely right!
You can say
            The child stays on the bike.

3.  The girls will no send the clever bully a baseball.
Good or bad, Jack?
>good
That was wrong.
It could be
            The girls won't send the clever bully a baseball.
```

4.3 Question Tutorial

The *question tutorial* generates statements and asks the student to change them to yes-no or wh-questions. The student determines the complexity of the statements the system creates, choosing whether or not to include such sentence characteristics as negatives and, if working at a high level, relative clauses.

****

[In this example, the student chose to work with yes-no questions at level 1 with vocabulary level 1.]

```
1.  The kid was smiling.
Make a yes-no question.
>was the kid smiling
```

[Note that the student does not have to have the correct capitalization or punctuation to have the answer judged correct.]

```
Good thinking!
Yes, the kid was smiling.

2.  The coats stay on the sled.
>**
```

[Here the student requests the complete answer by typing **.]

```
Do the coats stay on the sled?
Yes, the coats stay on the sled.

3.  Pat was smiling.
```

[Now the student will request one word of the answer at a time by typing a *. The italicized portion of each line was typed by the system.]

```
>Was *
>Was Pat *
>Was Pat smiling?
Yes, Pat was smiling.

4.  The boys played.
>did the boys played
>Did the boys play
Good thinking, Lyn!
Yes, the boys played.

5.  The toy was new.
>hint
Type in a YES-NO question.
>was the toy new?
Super!
Yes, the toy was new.
```

[In the following transcript, the student chose to work with wh-questions on the object of the sentence, at level 4 with vocabulary level 4. Notice that the congratulatory messages are more complex than those given above.]

```
1.  The adults brought Ms. Brown a dog.
Make a WH-question about the object.
>what did the adults bring Ms. Brown?
A worthy effort, Lyn!

2.  The boys loved a woman.
>**
```

[The student requested that the whole answer be displayed. Since she did not complete the answer herself, no congratulation is given.]

```
>Who did the boys love?

3.  The old teacher will mail Mr. Green a letter.
>*
>What will the *
>What will the old teacher mail Mr. Green
Prodigious!

4.  The old fire chief got the woman a pet.
>what did *
>What did the *
>What did the old *
>What did the old fire *
>What did the old fire chief get the worman
>What did the old fire chief get the woman
Superior!

5.  Jane wrote Andi a letter.
>hint
Type in a WH question.
>what did Jane wrote Andi?
>What did Jane *
>What did Jane write andi
I couldn't have done it better myself!
```

[The following examples were taken from a session in which the student wanted to work on wh-questions about the subject of the sentence, at level 1 with vocabulary level 4.]

```
1.  Kevin was smiling.
Make a WH-question about the subject.
>who was smiling?
Super!
```

```
2.  The boy wasn't skipping.
Make a WH-question about the subject.
>who wan't skipping?
>Who wasnt
>Who wasn't skipping
Super!

3.  Nan didn't lock the gate.
Make a WH-question about the subject.
>who didn't lock the gate?
Nice!

4.  The poodle wasn't barking.
>*
>What wasn't barking?
Super!

5.  Dr. Glinski was riding the bike.
>who was **
Who was riding the bike?
```

4.4 Subject-Verb Agreement Tutorial

The *subject-verb agreement tutorial* asks the student to change singular subjects to plural, and vice versa, and to make the appropriate morphological adjustments to nouns and verbs. The student may also control the syntactic complexity of sentences in this tutorial.

Here is an example of the interaction in the singular-plural tutorial where the difficulty level is set at 4 and the vocabulary level is set at 2:

```
**************************************************************************

SINGLE/PLURAL TUTORIAL - You will change sentences.

Now choose the kind of sentences you want.
Type choices to see additional choices; type go when finished.

transitive-sentence-with-indirect-object
                                predicate-adjective-sentence
predicate-adverb-sentence       predicate-nominal-sentence
negative-sentence               relative-clause
>>predicate-nominal-sentence
>>go

Thank you.  Now we can start.
```

```
Here is a sentence with a singular subject:

     1. The cat has been stolen by the child who Mr. Green tripped.

Now type it with a plural subject:
>The cats has been
```

[The student typed part of the sentence followed by a return. The tutorial only accepts as much of the sentence as is correct. Thus it echoes back only "The cats". Now the student doesn't know what was wrong with the other part that s/he typed so s/he types "hint" to get some clue.]

```
>The cats hint
```

[The tutorial gives the student the information that the verb might also have to be changed and also tells the student what the verb is. This is only supposed to be a clue to keep the student thinking. It is not intended to give the answer away.]

```
The verb, ''has been stolen'', may also have to be changed.
>The cats have been stolen by
```

[The student tries changing the verb and then types a return to see if that was correct. The student then realizes that the rest of the sentence remains unchanged, and types ** to save typing by having the tutorial complete the sentence.]

```
>The cats have been stolen by **
the child who Mr. Green tripped.
```

The singular-plural tutorial best demonstrates the Help system, showing how the tutorial can use its knowledge about the sentence to provide the student with information that is relevant to that tutorial. In this case, it is able to tell the student what the verb is and can remind the student that the verb may have to be changed so that there is number agreement. This type of knowledge concerns syntactic structure. The next example shows that the Help system can also use its knowledge about the morphological information in the dictionary.

```
Here is a sentence with a plural subject:

     2. The adults have been baking Jay a cake.

Now type it with a singular subject:
>hint
The subject is: the adults
>help
To singularize ''adults'', drop the ''s'' from the end of the word.
>The adult has been baking Jay a cake.
Terrific!
```

4.5 Pattern Tutorial

The *pattern tutorial* was created at the specific request of teachers at the Boston School for the Deaf. ILIAD generates sentences according to the five sentence pattern types in the Rhode Island School for the Deaf curriculum, which is in use at the Boston School. The student can determine the syntactic complexity of the sentences and also the number of different pattern types that will be generated within one session. At the Hint command, ILIAD displays the sentence broken into constituent parts.

The pattern tutorial demonstrated that it is a relatively simple task to produce a tutorial which conforms to the specific needs of students. The tutorial was tailored so as to use the terminology with which the students are familiar. This tutorial also makes use of ILIAD's detailed knowledge of the structure of the sentences it generates. Thus it is possible for the tutorial to give the breakdown of the sentence to aid the student in identifying the pattern.

The following example shows the interaction in the pattern tutorial:

```
**************************************************************************

PATTERN TUTORIAL  - you will identify sentence patterns.

Choose which sentence patterns you want.
1 - PATTERN 1    N  V
2 - PATTERN 2    N  V  N
3 - PATTERN 3    N  LV(be)  Adj
4 - PATTERN 4    N  LV(be)  N
5 - PATTERN 5    N  LV(be)  when/where

1, 2, 3, 4, 5 or All ?  all
```

```
1.  The banana will be sweet.
What type is that, Jack?
> 4

No, Jack, look at it again.
The banana  /   will be  /   sweet.

Try again, Jack.
>3
Nice going!

2.  The man was unhappy.
What type is that, Jack?
>3
Well done!

3.  Bette will make up the story.
What type is that?
>hint

The sentence parts are:
  Bette  /   will make up  /   the story.

>patterns
```

[If the student forgets what the sentence patterns are, s/he can see them again by typing "patterns".]

```
Remember the sentence patterns.
PATTERN 1    N  V
PATTERN 2    N  V  N
PATTERN 3    N  LV(be)  Adj
PATTERN 4    N  LV(be)  N
PATTERN 5    N  LV(be)  when/where

Bette will make up the story.
What type is that?
>2
Absolutely right!

4.  Jay hugged Mary.
What type is that?
>5
No, Jack, look at it again.
Jay  /   hugged  /   Mary.

Answer it again, Jack.
>2
Nice going!
```

50

```
5.  Bill was kind.
What type is that, Jack?
>3
Good going!

6.  Kim tripped the kids.
What type is that, Jack?
>2
Are you a genius?

7.  The man will be happy.
What type is that, Jack?
>5
WRONG!
The man  /   will be  /   happy.

Try again, Jack.
What type is that, Jack?
>1

Wrong again.
The correct answer is PATTERN 3
```

[Since the student got it wrong twice the correct answer is given.]

```
8.  The old toy is on the cement.  What type is that?  >score
```

[The student can ask to see his/her score so far by typing "score".]

```
Your score in the PATTERN TUTORIAL is as follows:

6 right.   1 wrong.   0 answers given by the computer.
7 total presented.
```

[The column "answers given by the computer" refers to situations where the student didn't respond at all within a certain amount of time at which point the system gives the student the right answer.]

4.6 Politeness Tutorial

The *politeness tutorial* is set up so that the student is presented with a situation in which some person wants somebody else to do something. This is followed by a request that might be used in that situation. The student has to identify the politeness level of that request. The various levels are BOSSY, STRONG, POLITE and VERY POLITE. For example, if one wanted Mary to open the door, some possible request forms are as follows:

BOSSY WAYS →	Open the door!
	You open the door!
STRONG WAYS →	Can you open the door?
	Will you open the door?
	Open the door, please.
	Please, open the door.
POLITE WAYS →	Please, can you open the door?
	Can you please open the door?
	Can you open the door, please?
VERY POLITE WAYS →	I would appreciate it if you would open the door.
	Would it be possible for you to open the door?

Of course the various politeness categories are not absolute, but they are designed to at least give the student a feel for different categories.

The politeness tutorial demonstrates that a tutorial can exercise a student in the functional aspects of language as opposed to the purely syntactic. It is designed to give the student a feel for the subtlety of the various ways requests can be made and the reality of using language to elicit some response from another person. This tutorial has a very large set of request forms which are applicable to the real world and which can easily be expanded to include various regional expressions and slang.

Here is an example of a transcript in which the student selected long sentences (level 4) and medium vocabulary (level 2).

```
**********************************************************************

POLITENESS TUTORIAL - You will judge the politeness of sentences.

Requests can be BOSSY, STRONG, POLITE or VERY POLITE.

Here are some requests.
Are they BOSSY, STRONG, POLITE or VERY POLITE?
Type b for BOSSY, s for STRONG, p for POLITE, v for VERY POLITE.
```

```
1. Greg wants a kid to help Dr. Johnson.
He says, "You help Dr. Johnson!"
Is he being BOSSY, STRONG, POLITE or VERY POLITE?
>very polite

VERY POLITE is way off, George.
Greg is being more direct.
Try again.

>strong
You can do better than that!
Greg is being BOSSY.
```

[Note that the messages given attempt to direct the student toward the right answer by indicating whether the original answer was close to the correct one or not. The same basic situation is used to generate exercises until the student requests a new basic sentence. This permits the student to see many different ways of framing the same request.]

```
2. Greg wants a kid to help Dr. Johnson.
He says, "I would be happy if you would help Dr. Johnson."
Is he being BOSSY, STRONG, POLITE or VERY POLITE?
>very polite
Will wonders never cease?

3. Greg wants a kid to help Dr. Johnson.
He says, "Would it be possible for you to help Dr. Johnson?"
Is he being BOSSY, STRONG, POLITE or VERY POLITE?
>polite
I couldn't have done it better myself!

4. Greg wants a kid to help Dr. Johnson.
He says, "Can you please help Dr. Johnson?"
Is he being BOSSY, STRONG, POLITE or VERY POLITE?
>new-sentence

Ok, we will make a new sentence.
First finish this one.
```

```
He says, "Can you please help Dr. Johnson?"
Is he being BOSSY, STRONG, POLITE or VERY POLITE?
>bossy

Shape up!
Greg is being POLITE.
```

[The tense and pronouns used in all parts of the interaction agree with the basic sentence that was generated. This illustrates that the tutorials can use detailed syntactic information about parts of the sentences produced by the generator.]

```
5. Nancy wanted the kid to get the sad lady a yellow apple.
She said, "It wouldn't be so hard to get the sad lady a
yellow apple now, would it?"
Was she being BOSSY, STRONG, POLITE or VERY POLITE?
>polite
I couldn't have done it better myself!

6. Nancy wanted the kid to get the sad lady a yellow apple.
She said, "Please, would it be possible for you to get
the sad lady a yellow apple?"

Was she being BOSSY, STRONG, POLITE or VERY POLITE?
>very polite
You got it!!!
```

4.7 Please Tutorial

The *please tutorial* presents the student with a request and requires the student to insert the word PLEASE in the appropriate places within that request. This tutorial also includes sentence forms which do not take PLEASE, to ensure that students realize that PLEASE cannot always occur at the beginning or end of requests. To minimize the amount of typing by the student, all s/he has to do is type a sequence of the characters "W" and "P"; W causes the next word in the sentence to be printed, P causes the word PLEASE to be printed. If the student makes an error, XXXX is printed and the student can try again. After the PLEASE has been inserted, the rest of the request is given, since there is no point in slavishly completing the request once the point where the PLEASE can be inserted has been found.

For sentences with modals there are three places where the word PLEASE can be inserted to make a request:

```
Please could you open the door?
Could you please open the door?
Could you open the door, please?
```

For ordinary commands there are two places PLEASE can be inserted:

```
Please open the door!
Open the door, please!
```

For longer or more round-about requests there is usually only one place that PLEASE can be inserted:

```
I would like you to open the door, please.
```

For some sentences PLEASE cannot be used at all:

```
You open the door!
```

Here is an example of one exercise:

```
1. Would it be possible for you to grease the bicycle?
```

[The student types WWWWWWWWWWP but sees the following line print out.]

```
>Would it be possible for you to grease the bicycle,. please?
There is 1 more way to add PLEASE to this request.
```

[Here the student just types "P", ILIAD types the rest of the request.]

```
>Please, would it be possible for you to grease the bicycle?
That's right!

2. Write the sad bullies a story!
```

[The student types P.]

```
>Please, write the sad bullies a story!
There is 1 more way to add PLEASE to this request.
```

[The student types WWWWWWP.]

```
>Write the sad bullies a story, please!
Prodigious!

3. I want you to tell the teacher a sad story.
>Please, I want you to tell the teacher a sad story.
There is 1 more way to add PLEASE to this request.
>I want you to tell the teacher a sad story, please.
Commendable!
```

```
4. Could you help?
>Please, could you help?
There are 2 more ways to add PLEASE to this request.
>Could you please help?
There is 1 more way to add PLEASE to this request.
>Could you help, please?
Will wonders never cease?
```

5. System Development Tools

5.1 Constraint Mechanism

The constraint mechanism in ILIAD is a highly flexible device for specifying which synspecs should be used to generate sentences for tutorial exercises. As choices are made, a context is created incorporating the various relationships between transformations; these relations have to do with which transformations can co-occur and which have dependency relationships since some transformations cannot apply unless other transformations have modified the underlying sentence structure. The rules which determine just how the specification of some syntactic choice will affect the context depend very much on the relationships among the choices previously made. The data base of synspecs contains all the possible syntactic choices and the relationships among them. The relationships between any two synspecs in the data base can to be expressed in terms of a few primitive relations. The constraint system "knows" about these primitives and determines the effects of making a syntactic choice. It is this part of the constraint mechanism that is easily extended or modified, since new primitives can be defined and new functions which know about these primitives can be integrated into the constraint mechanism.

5.1.1 Relationships between Synspecs

There are certain inherent and desired relationships that exist between synspecs. For example, certain synspecs such as passive-sentence and intransitive-sentence are mutually exclusive while others have more complicated relationships involving several synspecs.

All the relationships between the synspecs have been worked out and detailed. This information is part of the definition of each synspec. Thus, each synspec has information as to what other synspecs it conflicts with, expects to be enabled with or involves in some way.

In order to specify these relationships in a concise and adequate way, a "language" was defined which had to be powerful enough to capture all the possible relationships that might arise between synspecs. This "language" must have enough power, in terms of the primitives that make it up, to enable all the necessary relationships that are required for the particular set of items to be expressed and at the same time it must be simple, so that those relationships can be "coded" into it without requiring a detailed knowledge of the internals of the constraint system.

The language is made up of the following three primitives:

1. *contradicts*

   X *contradicts* (Y_1, Y_2, ... Y_n) is defined as meaning that synspec X cannot be chosen together with any of the synspecs (Y_1, Y_2, ... Y_n).

   e.g., yes-no-question *contradicts* (question-on-object question-on-subject)

   because a question can only be of one type.

   This does *not* imply that for each item Y_i for $1 \leq i \leq n$ there is a corresponding relationship in the data base of the form Y_i *contradicts* X for each i $1 \leq i \leq n$. That is, an item X can contradict another item Y without it necessarily being the case that Y contradicts X. However, in most cases this will not be the case and items will mutually contradict each other

2. *allows*

   X *allows* (Y_1, Y_2, ... Y_n) means that if synspec X is chosen then each synspec Y_i $1 \leq i \leq n$ is now a possible choice to be offered.

   e.g., passive-sentence *allows* agent-deletion

   If passive is chosen then it is meaningful to specify that the sentence should have agent deletion, making the difference between;

   The cookie was eaten by the boy
   The cookie was eaten.

   The possibility of agent-deletion must be presented only after it has been specified that the sentence is to be passive, but agent-deletion does not have to be chosen merely because it is a possibility.

3. *requires*

   X *requires* (Y_1, Y_2, ... Y_n) which means if synspec X is chosen then at least one of the synspecs Y_i $1 \leq i \leq n$ must be chosen.

   e.g., question *requires* (question-on-object, question-on-subject, yes-no-question)

5.1.2 Action of the Constraint Mechanism

The constraint mechanism starts off with a small subset of items that can be chosen at the top

level. As choices are made new choices are presented to the student and some other choices are removed from the student's range of choices. When the student has finished specifying his/her choices, the system might make some further choices depending on the set of options remaining. The sentence generating system will make choices in the same way as the student, using the same constraint mechanism, so that conceptually there is no difference between the student and the sentence generator making various specifications. All the constraint mechanism is concerned about is that valid choices are made, but who chooses them is irrelevant.

The program works with three sets:

- *CHOSEN*, which includes the choices chosen so far.
- *PRESENTED*, which includes all the choices that can be chosen at some point.
- *EXCLUDED*, which includes all the choices that can no longer be chosen.

When a new choice is made it is added to the set CHOSEN and removed from the set PRESENTED. Any choices that are now able to be presented as a result of the new choice are added to the set PRESENTED. Any choices that are now disallowed for any reason as a result of the new choice, are removed from PRESENTED (if they were in PRESENTED) and added to the set EXCLUDED.

5.1.3 Efficiency Considerations

The total number of synspecs that are used by the constraint mechanism is about 250. Since this is a fairly large set it is important that the constraint mechanism be efficient in terms of the amount of work it has to do to compute the state of the world after a synspec is chosen. The access methods for the data base of synspecs and their relationships must be fast and efficient. These efficiency goals have been achieved. The amount of work done after each choice is not dependent on the size of the data base of synspecs.

5.2 The Syntactic Playground

As one side effect of the development of the generative system, we have built a debugging environment called the syntactic playground in which a linguist or programmer can develop and test various components of the sentence generator. This environment is more useful than the tutorials in testing syntactic hypotheses and exploring the power of the generator. Using the playground debugging tools, dictionary entries, transformations, implications, and synspecs can be created,

edited, and saved using interactive routines that ensure the correct format of those data types. It is also possible to give commands to activate synspecs using the same interface that the tutorials use to the generator, or to set constraints "by hand". Flexibility is provided to control the generation process in a number of ways facilitating the development and testing of the generator.

The full power of the InterLISP system is available to the playground user as well as the specific commands designed for the interface. Thus a base tree can be edited directly, as can any version of the tree during the derivation process. Transformations can also be "broken" like functions so that when a transformation is about to be tried the generator goes into a "break" and conducts an interactive dialogue with the user who can then control the matching of the Structural Description, examine the results of the match, allow (or prohibit) the application of the Structural Change, edit the transformation and try it again, and perform many of the operations that are available in the playground. In addition to the break package, there is a trace option which prints a variety of information during the production of a sentence such as the constraints selected by the system, the words chosen for the base tree, the transformations which are attempted and whether they succeed or fail. The playground has proved to be a powerful tool for exploring and demonstrating the interaction of various rules and the efficacy of the whole generation package. A complete list of the commands that are available in the Playground is given in "ILIAD Data Base Reference".

6. Microcomputer Implementation

The MicroILIAD generation system has two subsystems, the utility section and the generation section. The utility section provides definitions and processing procedures for a variety of data structures used by the generation section. This has no analogue in the InterLISP version of ILIAD, since InterLISP provides these functions in the language itself. The generation section corresponds to the generation part of ILIAD, and includes a syntactic base component, a semantic component, and a transformational component. Two different semantic components have been implemented to test different methods of semantic representation. The tutorials have not yet been converted to Pascal.

6.1 Utility Section

A major design consideration for MicroILIAD has been to make use of existing InterLISP ILIAD code where possible, in order to ensure compatibility between the systems and to reduce the programming effort as modifications are made. To provide for this capability, a list processing system has been implemented in Pascal which handles most of the functions normally provided by a small InterLISP system but which has the function definitions in Pascal (rather than using a separate interpreter) to allow intermixing of InterLISP-like and Pascal operations. Thus all the InterLISP ILIAD data structures (alphabetic atoms, small integers, and arbitrary structures built from these) can be represented, although not all the vast collection of InterLISP functions have been defined in the Pascal system.

As the MicroILIAD system evolved, it became clear that the time and memory requirements for having a system with lists as the primary data structure would be prohibitive, so other data structures have been designed. The trees used throughout the system for representing sentences are stored efficiently using Pascal records linked together. To minimize space used by large collections of data (such as the transformations and the dictionary), an extremely compact coding has been developed where most of the significant data objects in the ILIAD system (node names, names of structural change functions, semantic categories, feature names and values, structural description elements) have been assigned single byte codes, alternately treated as small integers (for subscript computations) and characters (for storage as elements of character strings). One of the major utility modules provides definitions of the various codes and other data structures and routines to manipulate these structures.

In order to retain some compatibility with the InterLISP ILIAD system, as new data structures have been used in MicroILIAD, these new structures have been designed to be isomorphic to the InterLISP ILIAD structures, and programs have been provided to convert between the two sets. It is now possible to take an InterLISP ILIAD tree and convert it to MicroILIAD form, and to take a transformation from the InterLISP ILIAD transformation list and convert it automatically to MicroILIAD form. These conversions allow checking of transformations by directly comparing the operation of an InterLISP ILIAD transformation applied to a certain base tree with the result of the converted transformation applied to the converted base tree. Similarly, a grammatical rule expressed in InterLISP form can be converted to internal MicroILIAD form. (Although these InterLISP grammatical rules are not actually used within the ILIAD system, they have been written for part of the base grammar, and provide a readable version of this grammar.)

6.2 Generation Section

The generation section has the same major tasks as in InterLISP ILIAD, but the division of labor among them is different. In MicroILIAD, generation of a syntactic base structure is separated from lexical insertion and semantic analysis. This allows backup for alternate choices of words to meet semantic constraints without the necessity of replacing already generated syntactic structures. The transformational component operates very similarly to the InterLISP version, including in both systems a transformational morphology component.

6.2.1 Base Component

The base component is the first generation module used in the course of generating a sentence. Its function is to start with a set of base constraints specifying information about the syntactic form to be generated and to create a base tree, with nodes where lexical insertion will occur indicated by markers (e.g **NOUN**, **VERB**, etc.).

The base constraints are stored in Pascal records of two types, clause level constraints (specifying for example whether the verb is transitive or intransitive) which in turn point to constraint records for the noun phrases associated with the clause (subject, direct object, etc.). The noun constraints give information such as presence or absence of adjectives, person and number, and whether the phrase is definite or indefinite. In the case of noun phrases containing relative clauses (not presently used), the noun constraints will point to the clause constraints of the relative clause.

The other input to the base component is the base grammar. The grammar is originally specified as a LISP data structure in the form of an association list, giving possible expansions for each type of node depending on the base constraints. Before use in MicroILIAD, it is converted by program FILEGRAM into a Pascal record, where each set of possible expansions for a given node is indexed by the code for that type of node. This allows much faster access to the grammatical rules for a node. Logic coded within the base component determines which of a set of expansions is appropriate for a given set of base constraints. When an expansion is chosen, the nodes to be generated by the expansion are linked into the base tree and in turn expanded by a recursive call to the basic tree building function.

The output of the base component is a tree with nodes and features the same as the base tree of InterLISP ILIAD, but the data representation is different, and the leaves of the tree contain markers specifying the part of speech rather than words. This tree is then passed to the semantic component which replaces the markers with words to create the final base tree.

6.2.2 Feature-Based Semantics

The first semantic system implemented was based on use of semantic features. It operates after the base tree has been generated, and finds words to fill in the nodes marked with part of speech markers (e.g. **NOUN**). The system does lexical insertion in such a way that each word inserted is compatible with the base structure and all previously inserted words.

Each word has two types of features. One feature set is the features belonging to the word itself. For example, a verb might be marked as transitive, or a noun as HUMAN. The other type of features corresponds to Katz & Fodor style selectional restriction. For example, a verb might be marked as requiring an animate subject. Each word has one set of self-features plus up to three sets of selectional restrictions. These are generally used in a conventional manner, such as selectional restrictions applying from the verb to the nouns in its case frame, but could potentially apply in the opposite direction (e.g. a noun might have restrictions on an associated verb for which it is an object). This would be useful for allowing MicroILIAD to deal with idioms, in which such restrictions on the verb that a particular noun can occur with are common (e.g. the noun "headway" can only freely occur after the verb "make").

Each feature is, in fact, encoded as a pair of items corresponding to the feature and its opposite

(e.g. ANIMATE, INANIMATE). A feature which has a certain definite value will be marked by both the presence of the feature and the absence of its opposite (e.g. ANIMATE, NOT INANIMATE), while an optional or unspecified feature will have both the feature and its opposite specified. Selectional restrictions are coded similarly, except that an unspecified selectional restriction is coded by having neither the feature nor its opposite (e.g. NOT ANIMATE, NOT INANIMATE). In this way, a selectional restriction can be tested by simply checking whether its features specified are a subset of the features of the item to which the restriction applies. Also, by reversing the "don't care" coding, the selectional restrictions can be merged (using a set intersection operation) with the original features to give the effect of "transfer features". (For example, the verb "bark" might be marked as requiring a dog as subject (ignoring seals for the sake of this example), so in the sentence "The animal barked", the selectional restriction would imply that "the animal" was, in fact, a dog.)

Each word in the dictionary is marked with its part of speech and its feature set, and words of the appropriate class and compatible features are randomly selected for insertion into the base tree. As the words are inserted, a combined feature representation (original features plus transfer features) is built for each word, so that all words will be compatible when lexical insertion is finished. The final output from the semantic component is the original base tree with the part of speech markers replaced by words, ready for processing by the transformational component.

6.2.3 KLONE-Based Generation

The description of the semantic design based on the KLONE language is given in section 3.2.3. This section describes the particular implementation used for MicroILIAD, including issues of the interaction between the base component and the semantic component. The KLONE-based generation system uses a somewhat different control structure than the feature-based semantics system. In the feature-based version, the base tree was generated initially, then semantics was called to find words to fill in nouns, verbs, and adjectives in the tree. In the KLONE version, generation is initiated by the KLONE system's interpreting a KLONE structure, which determines both the base constraints and the words to eventually be inserted in the base tree. Then, the base constraints are passed to the base component to generate the base tree, and finally the KLONE-selected words are inserted in the tree. This organization allows a higher level of interaction between syntax and semantics while still maintaining some degree of modularity. The KLONE system does not do direct generation of base trees; instead, generation is mediated by the same base constraints used in the

previous system, so the KLONE semantics needs to know about base constraints but not about the grammar.

At present, the KLONE network on which generation is based is a unitary network which contains both general knowledge and specific instances of scenarios. A specific action node is selected for verb generation, and this node contains fillers for the various roles (actor, object, etc.), which are then used to generate the associated noun phrases. Most nodes contain a printable name by which they can be referenced. During generation, this name is selected as one of the words to be inserted in the base tree in the appropriate location. The generator allows a reference to be either to the specific node in the scenario (e.g. "Joe"), or to a node on the SUPERC chain (e.g. "the child"). There is a rule of thumb that no more than 2 SUPERC links can be traversed to find a node governing a print-name (to avoid, for example, "the human"), and no node with a nonprintable name (e.g. *PHYSOBJ*) will be used during generation.

As the KLONE structure is traversed, syntactic information also stored in the network is checked in order to set base constraints (e.g. if a node has part-of-speech *PROPERNOUN*, then base constraints will be set to skip generation of adjectives and determiners for this node) and to determine that the node has the correct part of speech for the portion of the base tree being generated (for example, a modifier role for a noun might be syntactically expressed as either an adjective or a prepositional phrase, and the part-of-speech links would distinguish these cases). Using the KLONE nodes which have been visited, the KLONE-based generator makes a table of parts of speech, grammatical relations (e.g. subject, direct-object) and the words selected for these categories. It then passes the base constraints which have been determined to the base component, takes the base tree output, and inserts the words from its table. The result is a lexicalized base tree ready for processing by the transformational component.

6.2.4 Transformational Component

The transformational component operates on the base tree after the semantics component has done lexical insertion, and produces a final tree ready to be passed to the sentence printing routine. The leaves of the transformed tree contain the final sentence form except for some details such as capitalization and punctuation which are done by the printing routine.

There are two classes of transformations, syntactic and morphological. The syntactic

transformations add, delete, or reorder constituents, while the morphological ones operate on the words at the leaves of the tree. The MicroILIAD transformations have the same information, although a different representation, as the InterLISP transformations described in Section 3.1.2

Each transformation has two major parts, the Structural Description and the Structural Change, plus a section for flags (for example, to indicate if the transformation is repeatable). The Structural Description and Structural Change are both represented as a sequence of codes. In the Structural Description, these codes are interpreted as node names, variables, or other objects specifying how the Structural Description is to match the tree. In the Structural Change, they specify functions to perform and parameters for the functions.

As with the base grammar for MicroILIAD, there are two forms for representation of the transformations. One form is a list structure, identical to the InterLISP representation, and, in fact, transferred directly from InterLISP ILIAD to MicroILIAD. The other form is a Pascal record containing the sequences of codes described above. In the second form, each transformation is indexed by number (to reduce the size of the transformation file by eliminating long names). The use of numeric indices and coded transformations reduces the size so that the present transformation list is all available in primary memory when MicroILIAD is run. A utility program, FILTRANS, is used to convert the LISP format transformations into MicroILIAD format.

During operation, the transformational component first attempts to match the Structural Description to an input tree, taking into account specifications of specific nodes, variables, features, and optional constituents. If the Structural Description is matched, the Structural Change is interpreted and applied to the tree, moving or modifying constituents as specified. When the ordered list of applicable transformations has been applied, the result is a final tree whose leaves are the words of the output sentence in the correct sequence, which are then printed by the output routine.

7. Dissemination

ILIAD has been disseminated in the form of reports and presentations to interested educators and designers of computer-based instructional systems. ILIAD has been disseminated via the ILIAD/Deafnet project in a more direct way allowing deaf children and adults to use the system via terminals in their homes or classrooms.

7.1 Reports and Presentations

The following reports and papers have been produced during the ILIAD project.

Technical Reports:

Kirk Wilson and Madeleine Bates, ILIAD: A Generative Computer System to Teach Language to the Deaf, Technical Progress Report, Sept., 1978 - April, 1979.

Kirk Wilson and Madeleine Bates, ILIAD: A Generative Computer System to Teach Language to the Deaf, Technical Progress Report, April, 1979 - Sept., 1979.

Kirk Wilson and Madeleine Bates, ILIAD: A Generative Computer System to Teach Language to the Deaf, Technical Progress Report, Sept., 1979 - March, 1980.

Kirk Wilson and Madeleine Bates, ILIAD: A Generative Computer System to Teach Language to the Deaf, Technical Progress Report, March, 1980 - Sept., 1980.

Kirk Wilson and Madeleine Bates, ILIAD: A Generative Computer System to Teach Language to the Deaf, Technical Progress Report, Sept., 1980 - March, 1981.

Kirk Wilson and Madeleine Bates, ILIAD Data Base Reference, Technical Report, Sept. 1981.

Papers:

Kirk Wilson and Madeleine Bates, "A Generative Computer System to Teach Language", Conference of the Association for the Development of Computer-based Instructional Systems, March 1979, San Diego, California.

Madeleine Bates and Kirk Wilson, "A Natural Language Micro-Computer System for English Instruction", Conference of the Society for Learning Technology, May, 1979, Washington, D.C.

Kirk Wilson, "Using computers in language and reading instruction.", Second Annual Massachusetts Office of Deafness Convention for Service Providers to Hearing Impaired Individuals, October 29, 1979, Framingham, Massachusetts.

Madeleine Bates and Kirk Wilson, "Project Report on ILIAD", Conference of the Association for Development of Computer-based Instructional Systems, March, 1980, Washington, D.C.

Madeleine Bates and Kirk Wilson, "Language instruction without pre-stored sentences", 1980, in *Proceedings of the 3rd Canadian Symposium on Instructional Technology*, Vancouver, British Colombia.

Madeleine Bates, Jack Beinashowitz, Robert Ingria and Kirk Wilson, "Generative Tutorial Systems", 1981, in *Proceedings of the 1981 Association for the Development of Computer-Based Instructional Systems Conference*, Atlanta, Georgia, March 3.

Kirk Wilson and Madeleine Bates, "Artificial intelligence in computer-based language instruction", 1981, in F.Withrow (Ed.) *Learning Technology and the Hearing Impaired*, Alexander Graham Bell Association for the Deaf, Washington, D.C.

Madeleine Bates and Robert Ingria, "Controlled Transformational Sentence Generation", 1981, in *Proceedings of the 1981 Annual Meeting of the Association for Computational Linguistics*, Stanford, California, July 1.

Kirk Wilson, *A Bibliography of English Language Development in Deaf Children and Adults*, 1981, Gallaudet College Press, Washington, D.C.

Theses:

Jack Beinashowitz, *A Constraint Based Interface which Protects the User from Making Incompatible Specifications*, M.A. Thesis, Department of Mathematics, 1981, Boston University.

Varda Shaked, *Representation of Nominal Concepts in Semantic Networks: Application for Generation of Predicate Nominal Sentences*, M.A. Thesis, Department of Mathematics, 1981, Boston University.

Presentations:

Presentations of ILIAD were given at the University of Massachusetts at Amherst, Columbia University, Gallaudet College, University of California Medical School at San Francisco, California State University at Northridge and the California School for the Deaf at Berkeley.

7.2 ILIAD/Deafnet Project

The ILIAD/Deafnet project was supported by the Handicapped Media Services and Captioned Films Program to introduce ILIAD to deaf children and adults and to bring deaf people into the design process of the ILIAD language instruction system. Since the ILIAD/Deafnet project was in direct support of the objectives of the main ILIAD project it is appropriate to indicate briefly in what ways ILIAD benefited from the input of deaf children and adults

Over the past two years the ILIAD system was introduced to (1) deaf adults using the Deaf Community Center Deafnet, (2) a class of deaf children at the Boston School for the Deaf, and, (3) during summers, eight deaf children of parent members of the Massachusetts Association for Deaf and Hard of Hearing Children. Users of ILIAD were taught how to use Hermes, an electronic message facility and the ILIAD message system, which allows students to send comments about ILIAD while within a tutorial (thereby avoiding the necessity to get out of the ILIAD tutorial program and into the Hermes message system to compose a message). In addition to messages received from the students regarding the positive and negative aspects of tutorials, the content of each tutorial session was recorded and reviewed to determine what kinds of problems students were having and which tutorials seemed to be most interesting (based on the frequency with which tutorials were selected and the number of items completed for each tutorial session).

The classroom at the Boston School for the Deaf included ten and eleven year old deaf children with immediate access to ILIAD via a terminal located in the classroom. ILIAD was extremely well received by the teacher coordinating the student training at the school. Two of the ILIAD tutorials were developed in response to teacher requests and a large number of teacher suggestions for tutorial designs have been catalogued for future implementation. A more complete review of the ILIAD/Deafnet project is available in ILIAD/Deafnet technical reports (O.E. Grant # G007904514, Paul Andereck: Project Officer).

8. Future of ILIAD

A substantial syntactic capability has been developed within the ILIAD language tutorial system and considerable design effort has been given to ILIAD semantics. With this linguistic foundation and our experience in the design and implementation of language tutorials for handicapped children, the ILIAD system has reached a point where a prototype system for dissemination may now be implemented on a low-cost microcomputer system. At the same time, a parallel effort should continue to expand the linguistics of ILIAD to incorporate an even broader range of syntactic structures, more features of KLONE semantics and an inference generation component.

With a prototype implementation should come field-testing and refinement of the user interface to ILIAD. The user interface involves both the student's degree of understanding and control in ILIAD tutorials as well as the teacher's use of the system as a supplement to classroom instruction. Determining how to introduce ILIAD concepts and interactive power to a language delayed student-user has been an on-going and unresolved problem. Considerable work remains to design an effective English or graphics-based metalanguage for controlling ILIAD tutorials. This work will include design of a simple interface for setting system constraints and for student-computer dialogues within tutorials. The student's ability to truly interact with the ILIAD system, particularly in using Help and Hint information, will be crucial to the system's usefulness and success.

A. Sentences Generated by ILIAD

1. Transitive Sentences

1. The bullies chased the girl.
2. What did the bullies do to the girl?
3. They chased her.
4. Who chased the girl?
5. The bullies chased her.
6. Who did they chase?
7. Whom did they chase?
8. They chased the girl.
9. How many bullies chased the girl?
10. Eight bullies chased the girl.
11. How many bullies chased her?
12. Eight bullies chased her.
13. Who got chased?
14. The girl got chased.
15. She was chased by the bullies.
16. The girl was being chased by the bullies.

2. Intransitive Sentences

1. What did the girl do?
2. She cried.
3. Who cried?
4. The girl cried.

3. Indirect Discourse

1. Dan said that the girl is sad.
2. Dan said that she is sad.
3. Who said that the girl is sad?

4. Transitive Sentence with Indirect Object

1. The generous boy gave a doll to the girl.
2. The generous boy gave the girl a doll.
3. The girl was given a doll.
4. A doll was given to the girl.
5. Who gave the girl a doll?
6. Who gave what to whom?
7. What did the generous boy give the girl?
8. He gave her a doll.
9. What did the generous boy give to the girl?

10. He gave a doll to her.
11. Who gave a doll to the girl?
12. Who gave the girl a doll?
13. Which boy gave the girl a doll?
14. The generous boy gave her a doll.
15. Which boy gave a doll to the girl?
16. The generous boy gave it to her.
17. How many dolls did the generous boy give the girl?
18. He gave her one doll.

5. Comparative Sentences

1. The soldier was better.
2. The gentleman will be more unhappy.
3. Alicia is hungrier than Jake.
4. The children were angrier than Andy.

6. Superlative Sentences

1. A policeman caught the nicest butterflies.
2. A sheepdog was the sickest pet.
3. The fire chief looks most generous.
4. The smartest man swore.
5. The oldest bulldog broke the dolls.

7. Sentences with Infinitives

1. The teacher wanted Kathy to hurry.
2. The gentleman promised the lady to close the door.
3. The girls were hard to ridicule.

8. Relative Clauses

1. Whoever embraced the kids will embrace the ladies.
2. The girl who was intelligent cheated the adults.
3. The woman who greased the tricycle mumbled.
4. The teacher who lost the bulldogs swears.

9. Negative Sentences

1. Kim won't help.
2. Claire didn't help.
3. The children won't shout.
4. Do not slap the poodles.
5. Do not cry.

10. Varieties of Quantifiers

1. No toy breaks.
2. Some excited boys kissed the women.
3. Some hungry people eat.
4. Two men cried.
5. Every new toy broke.
6. Not every man slips.
7. The boy won't give the dogs any oranges.
8. The girl doesn't see any cats.
9. The old men didn't tell the boys any thing.
10. The girl didn't love any body.

11. Varieties of Pronouns

1. Bette is the sad one.
2. Gloria is the happy one.
3. Kevin is the saddest.
4. Kathy is the most cheerful.
5. Varda liked the sweet apple.
6. Varda liked the sweet one.
7. The child tore the skirt.
8. That child tore the skirt.
9. This child tore the skirt.
10. The child tore this skirt.
11. The child tore that skirt.
12. This child tore that skirt.
13. That child tore that skirt.

12. THERE Sentences

1. There were some toys in the dirt.
2. There were no toys in the dirt.
3. There weren't any toys in the dirt.
4. There wasn't any one on the bike.

13. -/DEAF Sentences

1. People no will eat.
2. The woman will no tell the replies.
3. The cookies were no eaten by the man.
4. People not will eat.
5. No the people did eat.
6. The people did eat no.
7. Bully kidnapped the girls.
8. Seven man told the people the reply.

9. Six excited person told the girls the answer.
10. The nice man was play with the dog.
11. The cats are hide.
12. The children will have tripping the woman.
13. The creatures have running away.
14. The balls have nice.
15. The dog have young.
16. The cats be run away.
17. Alicia be slipped.
18. What the men give the dogs?
19. What the girls tell the people?

14. -** Ungrammatical Sentences

1. Played the games who?
2. What did lose Greg?
3. Played the old teacher the games.
4. Surprised who the youngsters?